3.6. MINIMUM-COST FLOW PROBLEM

In the preceding section we considered the problem of sending the maximum amount of flow value from source s to sink t in a network with specified edge capacities. There was no cost involved. We will now consider the problem of sending a specified amount of flow value θ (which we will call *target flow*) from s to t in a network G in which every edge (i, j) has a capacity c_{ij} as well as a nonnegative cost d_{ij} associated with it. The "cost" d_{ij} of sending a unit flow through edge (i, j) may represent the driving time, gas consumption, or any other measure depending on the application.

Obviously, if θ is greater than the maximum flow value $\mathcal{F}$ from s to t, no solution is possible. If θ is smaller than or equal to $\mathcal{F}$, then, in general, many different flow patterns will result in establishing a flow value θ from s to t in G. Our purpose is to find that flow pattern which minimizes the total cost. This problem is called the *minimum-cost flow problem* (or the *min-cost flow problem*) and may be stated more formally as follows:

To minimize

$$\sum_{(i,j)} d_{ij} f_{ij} \tag{3-5}$$

under the following constraints:

1. For every edge (i, j) in G,

$$0 \leqslant f_{ij} \leqslant c_{ij} \tag{3-6}$$

2. There is a specified node s in G, called the *source*, for which

$$\sum_i f_{si} - \sum_i f_{is} = \theta \tag{3-7}$$

where the summation $\Sigma_i f_{ij}$ is over all incoming edges to node j and $\Sigma_j f_{ij}$ is over all outgoing edges from node i.

3. There is another specified node t in G, called the *sink* or *destination*, for which

$$\sum_i f_{ti} - \sum_i f_{it} = -\theta \tag{3-8}$$

4. For each of the remaining nodes j, called *intermediate* nodes,

$$\sum_i f_{ji} - \sum_i f_{ij} = 0 \tag{3-9}$$

Observe that if all $c_{ij} = 1$, and $\theta = 1$, that is, if there is no edge-capacity

constraint, the problem is reduced to one of finding a *cheapest* (shortest) path from s to t with d_{ij} as the cost (distance or weight). In fact, even if the edge capacity constraints are imposed, the method of shipping θ amount of commodity along the cheapest path will work if the smallest edge capacity along this path is not less than θ. On the other hand, if the target flow value θ is greater than the smallest edge capacity along the cheapest path, the rest of flow has to be shipped through more expensive paths. This observation suggests that we can solve the min-cost flow problem by repeated application of a shortest-path algorithm, as follows.

First, we find the shortest-path (least-cost path) from s to t using a shortest-path algorithm (from Section 3.3). Then we send as many units of flow as possible along this path. If we have achieved the target flow value θ, we are through; otherwise, we modify (to be discussed shortly) the network, taking into account the flow pattern obtained so far. In this modified network, once again we find the shortest path from s to t and send as many units of flow as possible. These two basic steps of finding the shortest path (in the modified network) and modifying the network (due to the added flow) are repeated alternately until either the specified target flow value θ is obtained or there is no path left from s to t (i.e., θ exceeds the maximum possible flow value $\mathcal{F}$ through G).

Network Modification

The *modified network* (also called the *incremental network*) with respect to a flow pattern in a network G is the network G^* with the same structure as G but with *modified capacities* c_{ij}^* and *modified costs* d_{ij}^* defined as follows:

If there is a nonzero flow f_{ji} through an existing edge (j, i), we have a (fictitious) edge (i, j) "usable" in the reverse direction with capacity f_{ji}; that is,

$$c_{ij}^* = f_{ji} \qquad \text{if} \quad f_{ji} > 0$$

The cost associated with this edge is $-d_{ji}$, because the edge (i, j) is usable only by virtue of a flow reduction (and hence cost reduction); that is,

$$d_{ij}^* = -d_{ji} \qquad \text{if} \quad f_{ji} > 0$$

Also, if there is a flow f_{ij} through an existing edge (i, j) which is less than its capacity c_{ij}, we can send additional flow through this unsaturated edge (i, j) in forward direction (provided, of course, that there is no flow f_{ji}). Thus

$$c_{ij}^* = c_{ij} - f_{ij} \qquad \text{if} \quad f_{ji} = 0 \quad \text{and} \quad c_{ij} > f_{ij}$$

The cost of sending 1 unit of additional flow through this unsaturated edge is the same as the original cost. That is,

$$d_{ij}^* = d_{ij} \qquad \text{if} \quad f_{ji} = 0 \quad \text{and} \quad c_{ij} > f_{ij}$$

Finally, since a saturated edge is not usable in the forward direction, we have

$$c^*_{ij} = c_{ij} - f_{ij} = 0 \qquad \text{if} \quad f_{ij} = c_{ij} \quad \text{and} \quad f_{ji} = 0$$

and

$$d^*_{ij} = \infty \qquad \text{if} \quad f_{ij} = c_{ij} \quad \text{and} \quad f_{ji} = 0$$

Note that we maintain throughout the constraint $f_{ij}f_{ji} = 0$, because we cannot have flows in opposing directions through any edge at the same time. All edges with no flows remain unchanged in the modified network G^*.

Busacker–Gowen's Algorithm

Now, having defined the modified network, we can express the *min-cost path-flow* algorithm as follows:

Algorithm 3-7(a): Outline of Busacker and Gowen's Min-Cost Flow Algorithm

```
begin
    G* ← G;
    while (flow value < θ) and (a path exists from s to t in G*) do
    begin
        find a shortest path P from s to t in G*;
        if no such path exists then exit;
        send additional flow along P until either P is saturated or the flow reaches θ;
        if P saturates then obtain a modified network G*
    end
end
```

This "incremental approach" of achieving larger and larger flows through paths of increasing costs was originally proposed by Busacker and Gowen [1961]. It is also called the *dual method* of solving the min-cost flow problem, because the first feasible solution obtained is an optimal solution. The following small example illustrates further details of this algorithm.

The five-node seven-edge network G shown in Fig. 3-24, has two labels on each edge. The first number is the cost of sending a unit flow and the second number is the capacity. We wish to find a minimum-cost flow of 25 units from s to t.

Iteration 1

The shortest (least-cost) path from s to t in Fig. 3-24 is $P = (s, a, c, t)$. The cost of sending 1 unit of flow through this path is $3 + 4 + 3 = 10$. The maximum flow possible through this path is $\min\{18, 15, 17\} = 15$ which is less than 25, the desired amount. Therefore, we establish a flow of 15 units through this path. Thus the first

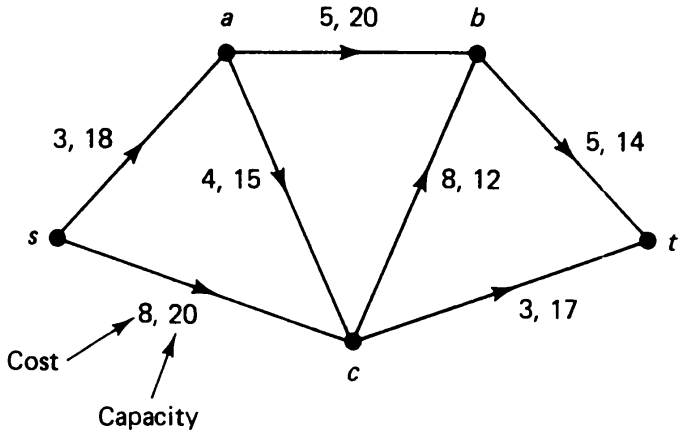

Figure 3-24 *Flow network with edge costs and capacities.*

flow pattern obtained is

$$f_{sa} = f_{ac} = f_{ct} = 15$$

We modify the network to take this flow pattern into account. The modified network is shown in Fig. 3-25. We go into the second iteration.

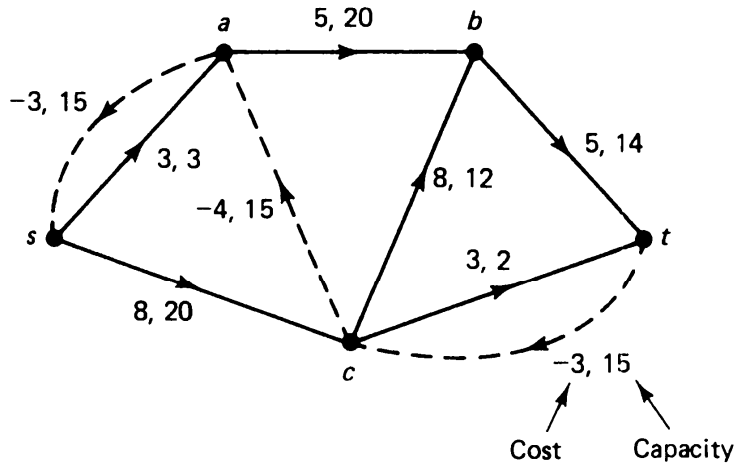

Figure 3-25 *Network after first modification.*

Iteration 2

In Fig. 3-25 the least-cost path from s to t is $P' = (s, c, t)$. The unit cost through this path is $8 + 3 = 11$. The maximum flow through this path is $\min\{20, 2\} = 2$. We ship 2 units through this path, thus achieving so far a total flow value of $15 + 2 = 17$ which is less than 25. The cumulative flow pattern obtained thus far is

$$f_{sa} = f_{ac} = 15, \qquad f_{ct} = 17, \qquad f_{sc} = 2$$

The modified network with respect to this flow pattern is given in Fig. 3-26. Next, we go into the third iteration.

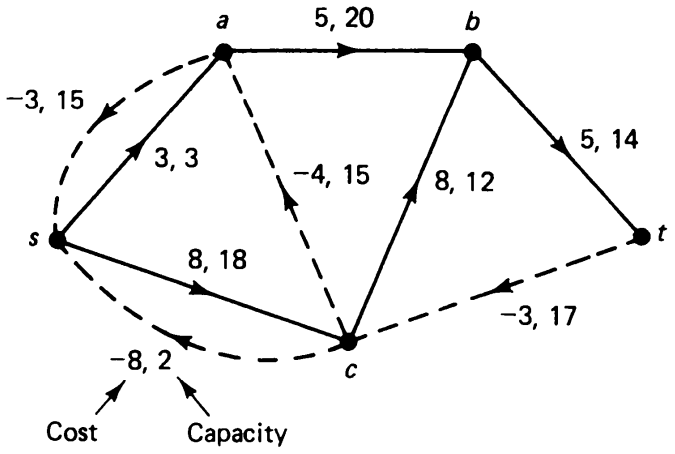

Figure 3-26 *Network after second modification.*

Iteration 3

In Fig. 3-26 the shortest path from s to t is $P'' = (s, a, b, t)$. The unit cost in P'' is $3 + 5 + 5 = 13$ and the maximum flow possible through P'' is $\min\{3, 20, 14\} = 3$. Sending 3 units through P'' results into a total flow value of $17 + 3 = 20$ and into the cumulative flow pattern of

$$f_{sa} = 18, \quad f_{ac} = 15 \quad f_{sc} = 2, \quad f_{ct} = 17 \quad f_{ab} = 3, \quad f_{bt} = 3$$

The modified network with respect to this flow pattern is given in Fig. 3-27.

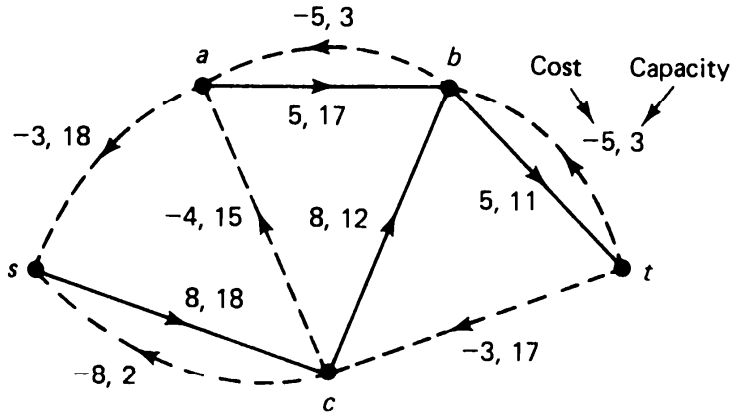

Figure 3-27 *Network after third modification.*

Iteration 4

The shortest path from s to t in Fig. 3-27 is $P''' = (s, c, a, b, t)$, with unit flow cost $8 - 4 + 5 + 5 = 14$ and maximum capacity $\min\{18, 15, 17, 11\} = 11$. Since we have achieved a flow value of 20 units, we require only 5 additional units of flow, which we can send through this path. This results into a total flow value of 25 units from s to t, and the final flow pattern is

$$f_{sa} = 18, \quad f_{sc} = 7 \quad f_{ab} = 8, \quad f_{ac} = 10 \quad f_{bt} = 8, \quad f_{ct} = 17$$

The total cost of this flow is

$$\sum_{(i,j)} d_{ij} f_{ij} = 3 \cdot 18 + 8 \cdot 7 + 5 \cdot 8 + 4 \cdot 10 + 5 \cdot 8 + 3 \cdot 17 = 281$$

Note that had we continued increasing the flow instead of stopping at 25 units, we would have obtained a flow of 31 units from s to t. After that, in the modified network, there would be no more usable paths left from s to t. Hence 31 units would be the value of the maximum flow through the network. Thus the maximum flow problem is a special case of the minimum-cost flow problem. However, as we will discuss later, this may be an inefficient way of solving the max-flow problem.

Now we will describe somewhat more formally Busacker and Gowen's min-cost flow algorithm just illustrated. We are given network G in which two vertices s and t are specified and the value of the target flow θ is also specified. Initially, we set the current value of the flow from s to t as 0. We also set the *modified network* G^* as G itself, to begin with. Moreover, we maintain a Boolean variable *stPath* to indicate the existence of a usable path from s to t. Initially, we set *stPath* to *true*.

The algorithm may now be described compactly as follows [the formal proof of the algorithm is left as an exercise (Problem 3-60)]:

Algorithm 3-7(b): Busacker and Gowen's Min-Cost Flow Algorithm

```
INITIALIZATION:
    CurrentFlow ← 0;
    G* ← G; {* modified network is G itself *}
    stPath ← true;

ITERATION:
    while (CurrentFlow < θ) and stPath do
    begin
        find shortest path P from s to t in G*;
        if no path exists then stPath ← false;
        if stPath then
        begin
            delta ← smallest usable edge capacity in P;
                {* maximum feasible flow through P *}
            if CurrentFlow + delta > θ then delta ← θ − CurrentFlow;
            for each edge (i, j) on P do {* establish a flow of amount delta
                through P by suitably increasing or decreasing flows through
                edges in P, and adjust costs *}
            begin
                if (i, j) is a forward edge {* i.e., capacity c*ij > 0 *} then
```

```
            begin
                increase flow f_ij through (i, j);
                if (i, j) is unsaturated then cost*(i, j) stays the same
                else cost*(i, j) ← ∞;
                if (i, j) is usable in backward direction then cost*(j, i) ← −cost(i, j)
            end
            else {* if (i, j) is a backward edge *}
            begin
                decrease flow f_ji through (j, i);
                adjust cost of edge (j, i) as in the forward case
            end;
            CurrentFlow ← CurrrentFlow + delta
        end {* for *}
    end {* stPath *}
end {* while *}
```

Shortest-Path Subprocedure

In implementing the min-cost flow algorithm just described, a crucial decision that we must make is in the choice of the shortest-path algorithm. Three shortest-path algorithms were described in Section 3.3, of which Floyd's algorithm is not suitable for our purpose, because it computes shortest paths between all pairs of nodes (at added computational cost), whereas we only need a shortest path from s to t. If the choice were between the Bellman–Moore algorithm and the Dijkstra's, we could select either. Both are fast. The following are some important considerations:

1. *Negative-weight edges.* In this min-cost flow algorithm, as we proceed with modified networks, many edge costs become negative. Dijkstra's algorithm does not normally work for networks with negative weights, as discussed in Section 3.3. However, the negative weights can be eliminated by an artificial device of adding suitable numbers to various edge costs (or weights) to keep them all nonnegative, without changing their relative costs. Thus the shortest path remains unaffected. This device was suggested independently by Iri [1969] and Edmonds and Karp [1972], and it enables one to use Dijkstra's algorithm for the min-cost flow problem. This does require some additional computations, thus making Dijkstra's algorithm more expensive than it would otherwise be. The Bellman–Moore algorithm, of course, has no difficulty in handling negative edges.

2. *One-to-one and one-to-all paths.* Recall that Dijkstra's algorithm could be terminated as soon as the destination node t received a permanent label, that is, as soon as the shortest path from s to t was obtained. In the Bellman–Moore algorithm, on the other hand, we must wait until the shortest paths from s to all other nodes have been obtained. Therefore, the latter is more suitable for

one-to-all shortest-path problems. It may, therefore, appear that Dijkstra's algorithm would have an advantage in the min-cost flow algorithm. However, due to negativity of edge weights, it turns out that in using Dijkstra's algorithm also we must wait until all nodes have been permanently labeled.

3. *Edge density.* It was mentioned in Section 3.3 that the PDM implementation of the Bellman–Moore algorithm outperformed Dijkstra's algorithm only when the network was sparse. Because of the added computational work in modifying Dijkstra's algorithm, our experience shows that the use of PDM procedure leads to overall shorter time for a typical min-cost flow problem, even when the network is complete (i.e., there is an edge between every pair of nodes, see Table 3.2).

4. *Worst-case input versus average-case input.* Although in a typical network the PDM implementation of the Bellman–Moore algorithm is faster (and more so for sparse networks), for a pathologically worst-case input (see Problem 3-15), it could take an exponential amount of time. Of course, a network such as the one shown in Problem 3-15 is not encountered in practice.

In light of these four considerations, we will implement the Busacker–Gowen min-cost flow algorithm [i.e., Algorithm 3-7(b)] using the (slightly modified) PDM procedure (Algorithm 3-2) for shortest paths. We will leave the use of the modified Dijkstra's procedure as an exercise.

Data Structure

The choice of data structure will also have a significant influence on the computation time. Since new edges are created as the network gets modified, it is not sufficient to have just a fixed number of edges (say m, in the forward-star form). We must keep a space for representing $2m$ edges, if the network originally has m edges. Because of this, and to be consistent with the MAXFLOW algorithm in the preceding section, we will use matrices to represent edge capacities (CAPA) and costs (COST). The reader is encouraged to make changes in order to use other data structures, such as the list of edges on forward star, which would be more efficient for very sparse and large networks.

Computer Implementation of Busacker–Gowen's Min-cost Flow Algorithm

The BUSACKER procedure finds a minimum-cost flow from a specified node S to another specified node T in an N-node network, represented by a cost matrix COST and a capacity matrix CAPA. The procedure closely follows the method outlined in Algorithm 3-7(b).