CHAPTER 14

Network Flow Problems

Many linear programming problems can be viewed as a problem of minimizing the "transportation" cost of moving materials through a network to meet demands for material at various locations given sources of material at other locations. Such problems are called *network flow problems*. They form the most important special class of linear programming problems. Transportation, electric, and communication networks provide obvious examples of application areas. Less obvious, but just as important, are applications in facilities location, resource management, financial planning, and others.

In this chapter we shall formulate a special type of linear programming problem called the *minimum-cost network flow problem*. It turns out that the simplex method when applied to this problem has a very simple description and some important special properties. Implementations that exploit these properties benefit dramatically.

1. Networks

A network consists of two types of objects: nodes and arcs. We shall let $\mathcal{N}$ denote the set of *nodes*. We let m denote the number of nodes (i.e., the cardinality of the set $\mathcal{N}$).

The nodes are connected by *arcs*. Arcs are assumed to be directed. This means that an arc connecting node i to node j is not the same as an arc connecting node j to node i. For this reason, we denote arcs using the standard mathematical notation for ordered pairs. That is, the arc connecting node i to node j is denoted simply as (i, j). We let $\mathcal{A}$ denote the set of all arcs in the network. This set is a subset of the set of all possible arcs:

$$\mathcal{A} \subset \{(i, j) : i, j \in \mathcal{N}, i \neq j\}.$$

In typical networks, the set $\mathcal{A}$ is much smaller than the set of all arcs. In fact, usually each node is only connected to a handful of "nearby" nodes.

The pair $(\mathcal{N}, \mathcal{A})$ is called a *network*. It is also sometimes called a *graph* or a *digraph* (to emphasize the fact that the arcs are directed). Figure 14.1 shows a network having 7 nodes and 14 arcs.

To specify a network flow problem, we need to indicate the supply of (or demand for) material at each node. So, for each $i \in \mathcal{N}$, let b_i denote the amount of material being supplied to the network at node i. We shall use the convention

The original version of this chapter was revised. An erratum to this chapter can be found at DOI 10.1007/978-1-4614-7630-6_26

R.J. Vanderbei, *Linear Programming*, International Series in Operations Research & Management Science 196, DOI 10.1007/978-1-4614-7630-6_14,

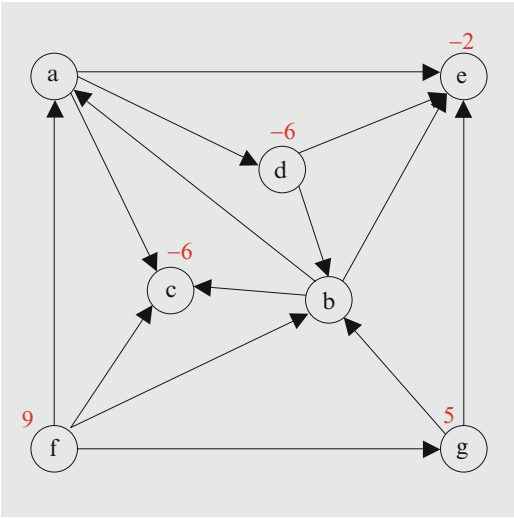


FIGURE 14.1. A network having 7 nodes and 14 arcs. The numbers written next to the nodes denote the supply at the node (*negative values* indicate demands; *missing values* indicate no supply or demand).

that negative supplies are in fact demands. Hence, our problem will be to move the material that sits at the supply nodes over to the demand nodes. The movements must be along the arcs of the network (and adhering to the directions of the arcs). Since, except for the supply and demand, there is no other way for material to enter or leave the system, it follows that the total supply must equal the total demand for the problem to have a feasible solution. Hence, we shall always assume that

$$\sum_{i \in \mathcal{N}} b_i = 0.$$

To help us decide the paths along which materials should move, we assume that each arc, say, (i, j), has associated with it a cost c_{ij} that represents the cost of shipping one unit from i to j directly along arc (i, j). The decision variables then are how much material to ship along each arc. That is, for each $(i, j) \in \mathcal{A}$, x_{ij} will denote the quantity shipped directly from i to j along arc (i, j). The objective is to minimize the total cost of moving the supply to meet the demand:

$$\text{minimize} \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij}.$$

As we mentioned before, the constraints on the decision variables are that they ensure flow balance at each node. Let us consider a fixed node, say, $k \in \mathcal{N}$. The total flow into node k is given by

$$\sum_{\substack{i: \\ (i,k) \in \mathcal{A}}} x_{ik}.$$

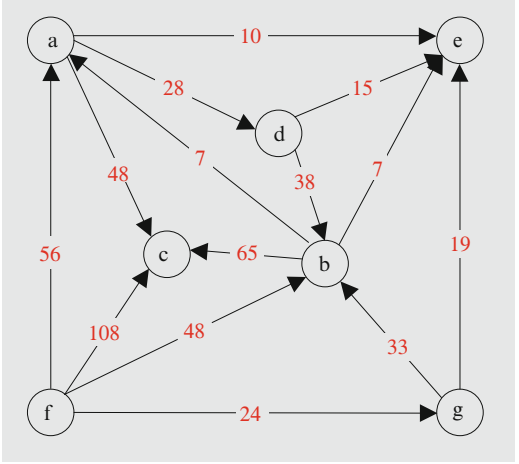


FIGURE 14.2. The costs on the arcs for the network in Figure 14.1.

Similarly, the total flow out from node k is

$$\sum_{\substack{j:\\(k,j)\in\mathcal{A}}} x_{kj}.$$

The difference between these two quantities is the net inflow, which must be equal to the demand at the node. Hence, the *flow balance constraints* can be written as

$$\sum_{\substack{i:\\(i,k)\in\mathcal{A}}} x_{ik} - \sum_{\substack{j:\\(k,j)\in\mathcal{A}}} x_{kj} = -b_k, \qquad k \in \mathcal{N}.$$

Finally, the flow on each arc must be nonnegative (otherwise it would be going in the wrong direction):

$$x_{ij} \geq 0, \qquad (i,j) \in \mathcal{A}.$$

Figure 14.2 shows cost information for the network shown in Figure 14.1. In matrix notation, the problem can be written as follows:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = -b \\ & x \geq 0, \end{array} \tag{14.1}$$

where

$$x^T = \begin{bmatrix} x_{ac} & x_{ad} & x_{ae} & x_{ba} & x_{bc} & x_{be} & x_{db} & x_{de} & x_{fa} & x_{fb} & x_{fc} & x_{fg} & x_{gb} & x_{ge} \end{bmatrix},$$

$$A = \begin{bmatrix} -1 & -1 & -1 & 1 & & & & & 1 & & & & & \\ & & & -1 & -1 & -1 & 1 & & & 1 & & & 1 & \\ 1 & & & & 1 & & & & & & 1 & & & \\ & 1 & & & & & -1 & -1 & & & & & & \\ & & 1 & & & 1 & & 1 & & & & & & 1 \\ & & & & & & & & -1 & -1 & -1 & -1 & & \\ & & & & & & & & & & & 1 & -1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ -6 \\ -6 \\ -2 \\ 9 \\ 5 \end{bmatrix},$$

$$c^T = \begin{bmatrix} 48 & 28 & 10 & 7 & 65 & 7 & 38 & 15 & 56 & 48 & 108 & 24 & 33 & 19 \end{bmatrix}.$$

In network flow problems, the constraint matrix A is called the *node–arc incidence matrix*.

The network flow problem differs from our usual standard form linear programming problem in two respects: (1) it is a minimization instead of a maximization and (2) the constraints are equalities instead of inequalities. Nonetheless, we have studied before how duality applies to problems in nonstandard form. The dual of (14.1) is

$$\begin{array}{ll} \text{maximize} & -b^T y \\ \text{subject to} & A^T y + z = c \\ & z \geq 0. \end{array}$$

Written in network notation, the dual is

$$\begin{array}{ll} \text{maximize} & -\displaystyle\sum_{i \in \mathcal{N}} b_i y_i \\ \text{subject to} & y_j - y_i + z_{ij} = c_{ij}, \qquad (i,j) \in \mathcal{A} \\ & z_{ij} \geq 0, \qquad (i,j) \in \mathcal{A}. \end{array}$$

Finally, it is not hard to check that the complementarity conditions (to be satisfied by an optimal primal–dual solution pair) are

$$x_{ij} z_{ij} = 0, \qquad (i,j) \in \mathcal{A}.$$

We shall often refer to the primal variables as *primal flows*.

2. Spanning Trees and Bases

Network flow problems can be solved efficiently because the basis matrices have a special structure that can be described nicely in terms of the network. In order to explain this structure, we need to introduce a number of definitions.

First of all, an ordered list of nodes $(n_1, n_2, \ldots, n_k)$ is called a *path* in the network if each adjacent pair of nodes in the list is connected by an arc in the network. It is important to note that we do not assume that the arcs point in any particular direction. For example, for nodes n_i and n_{i+1}, there must be an arc in the network. It could run either from n_i to n_{i+1} or from n_{i+1} to n_i. (One should think about one-way roads—even though cars can only go one way, pedestrians are allowed to walk along the path of the road in either direction.) A network is called *connected* if there is a path connecting every pair of nodes (see Figure 14.3). For the remainder of this chapter, we make the following assumption:

Assumption. The network is connected.

For any arc (i,j), we refer to i as its *tail* and j as its *head*.

A *cycle* is a path in which the last node coincides with the first node. A network is called *acyclic* if it does not contain any cycles (see Figure 14.4).

A network is a *tree* if it is connected and acyclic (see Figure 14.5). A network $(\tilde{\mathcal{N}}, \tilde{\mathcal{A}})$ is called a *subnetwork* of $(\mathcal{N}, \mathcal{A})$ if $\tilde{\mathcal{N}} \subset \mathcal{N}$ and $\tilde{\mathcal{A}} \subset \mathcal{A}$. A subnetwork $(\tilde{\mathcal{N}}, \tilde{\mathcal{A}})$ is a *spanning tree* if it is a tree and $\tilde{\mathcal{N}} = \mathcal{N}$. Since a spanning tree's node set coincides with the node set of the underlying network, it suffices to refer to a spanning tree by simply giving its arc set.

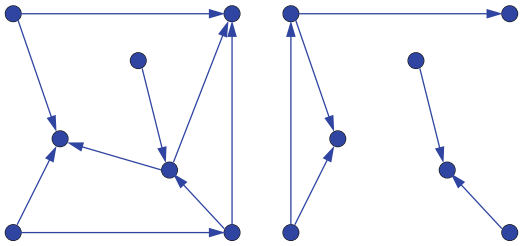


FIGURE 14.3. The network on the *left* is connected whereas the one on the *right* is not.

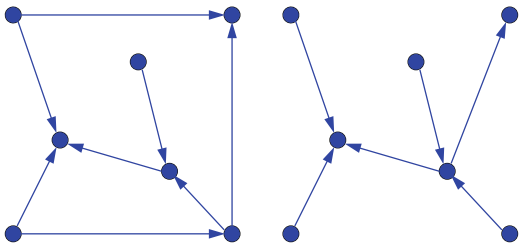


FIGURE 14.4. The network on the *left* contains a cycle whereas the one on the *right* is acyclic.

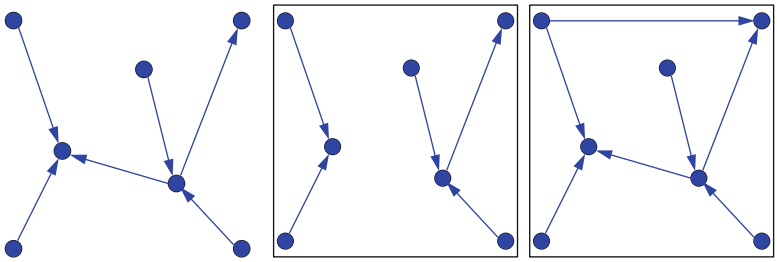


FIGURE 14.5. The network on the *left* is a tree whereas the two on the *right* not—they fail in the first case by being disconnected and in the second by containing a cycle.

Given a network flow problem, any selection of primal flow values that satisfies the balance equations at every node will be called a *balanced flow*. It is important to note that we do not require the flows to be nonnegative to be a balanced flow. That is, we allow flows to go in the wrong direction. If all the flows are nonnegative, then a balanced flow is called a *feasible flow*. Given a spanning tree, a balanced flow that assigns zero flow to every arc not on the spanning tree will be called a *tree solution*. Consider, for example, the tree shown in Figure 14.6. The numbers shown on the arcs of the spanning tree give the tree solution corresponding to the supplies/demands shown in Figure 14.1. They were obtained by starting at the

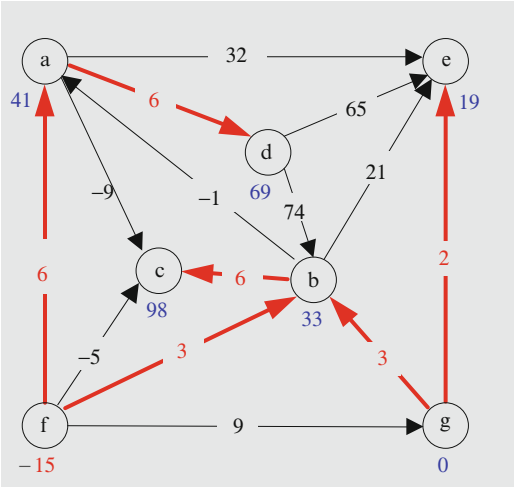


FIGURE 14.6. The *fat arcs* show a spanning tree for the network in Figure 14.1. The numbers shown on the arcs of the spanning tree are the primal flows, the numbers shown next to the nodes are the dual variables, and the numbers shown on the arcs not belonging to the spanning tree are the dual slacks.

"leaves" of the tree and working "inward." For instance, the flows could be solved for successively as follows:

$$
\begin{aligned}
&\text{flow bal at d:} && x_{\text{ad}} = 6, && \\
&\text{flow bal at a:} && x_{\text{fa}} - x_{\text{ad}} = 0 && \Longrightarrow \quad x_{\text{fa}} = 6, \\
&\text{flow bal at f:} && -x_{\text{fa}} - x_{\text{fb}} = -9 && \Longrightarrow \quad x_{\text{fb}} = 3, \\
&\text{flow bal at c:} && x_{\text{bc}} = 6, && \\
&\text{flow bal at b:} && x_{\text{fb}} + x_{\text{gb}} - x_{\text{bc}} = 0 && \Longrightarrow \quad x_{\text{gb}} = 3, \\
&\text{flow bal at e:} && x_{\text{ge}} = 2. &&
\end{aligned}
$$

It is easy to see that this process always works. The reason is that every tree must have at least one leaf node, and deleting a leaf node together with the edge leading into it produces a subtree.

The above computation suggests that spanning trees are related to bases in the simplex method. Let us pursue this idea. Normally, a basis is an invertible square submatrix of the constraint matrix. But for incidence matrices, no such submatrix exists. To see why, note that if we sum together all the rows of A, we get a row vector of all zeros (since each column of A has exactly one $+1$ and one -1). Of course, every square submatrix of A has this same property and so is singular. In fact, we shall show in a moment that for a connected network, there is exactly one redundant equation (i.e., the rank of A is exactly $m - 1$).

Let us select some node, say, the last one, and delete the flow-balance constraint associated with this node from the constraints defining the problem (since it is redundant anyway). Let's call this node the *root node*. Let $\tilde{A}$ denote the incidence matrix A without the row corresponding to the root node (i.e., the last row), and let $\tilde{b}$ denote the supply/demand vector with the last entry deleted. The most important property of network flow problems is summarized in the following theorem:

THEOREM 14.1. *A square submatrix of $\tilde{A}$ is a basis if and only if the arcs to which its columns correspond form a spanning tree.*

Rather than presenting a formal proof of this theorem, it is more instructive to explain the idea using the example we've been studying. Therefore, consider the spanning tree shown in Figure 14.6, and let B denote the square submatrix of $\tilde{A}$ corresponding to this tree. The matrix B is invertible if and only if every system of equations of the form

$$Bu = \beta$$

has a unique solution. This is exactly the type of equation that we already solved to find the tree solution associated with the spanning tree:

$$Bx_{\mathcal{B}} = -\tilde{b}.$$

We solved this system of equations by looking at the spanning tree and realizing that we could work our way to a solution by starting with the leaves and working inward. This process amounts to a permutation of the rows and columns of B to get a lower triangular matrix. Indeed, for the calculations given above, we have permuted the rows by P and the columns by Q to get

$$PBQ^T = \begin{array}{c} \\ \text{d} \\ \text{a} \\ \text{f} \\ \text{c} \\ \text{b} \\ \text{e} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{(a,d)} & \text{(f,a)} & \text{(f,b)} & \text{(b,c)} & \text{(g,b)} & \text{(g,e)} \end{array} \\ \begin{bmatrix} 1 & & & & & \\ -1 & 1 & & & & \\ & -1 & -1 & & & \\ & & & 1 & & \\ & & 1 & -1 & 1 & \\ & & & & & 1 \end{bmatrix} \end{array}.$$

The fact that B is invertible is now immediately apparent from the fact that the permuted matrix is lower triangular. In fact, it has only $+1$'s and -1's on the diagonal. Therefore, we can solve systems of equations involving B without ever having to do any divisions. Also, since the off-diagonal entries are also ± 1's, it follows that we don't need to do any multiplications either. Every system of equations involving the matrix B can be solved by a simple sequence of additions and subtractions.

We have shown that, given a spanning tree, the submatrix of $\tilde{A}$ consisting of the columns corresponding to the arcs in the spanning tree is a basis. The converse direction (which is less important to our analysis) is relegated to an exercise (see Exercise 14.12).

Not only is there a primal solution associated with any basis but also there is a dual solution. Hence, corresponding to any spanning tree there is a dual solution.

The dual solution consists of two types of variables: the y_i's and the z_{ij}'s. These variables must satisfy the dual feasibility conditions:

$$y_j - y_i + z_{ij} = c_{ij}, \qquad (i,j) \in \mathcal{A}.$$

By complementarity, $z_{ij} = 0$ for each (i,j) in the spanning tree $\mathcal{T}$. Hence,

$$y_j - y_i = c_{ij}, \qquad (i,j) \in \mathcal{T}.$$

Since a spanning tree on m nodes has $m-1$ arcs (why?), these equations define a system of $m-1$ equations in m unknowns. But don't forget that there was a redundant equation in the primal problem, which we associated with a specific node called the root node. Removing that equation and then looking at the dual, we see that there is not really a dual variable associated with the root node. Or equivalently, we can just say that the dual variable for the root node is zero. Making this assignment, we get m equations in m unknowns. These equations can be solved by starting at the root node and working down the tree.

For example, let node "g" be the root node in the spanning tree in Figure 14.6. Starting with it, we compute the dual variables as follows:

$$\begin{aligned}
& & y_{\mathrm{g}} &= 0, & & \\
\text{across arc (g,e):} & & y_{\mathrm{e}} - y_{\mathrm{g}} &= 19 & \Longrightarrow \quad & y_{\mathrm{e}} = 19, \\
\text{across arc (g,b):} & & y_{\mathrm{b}} - y_{\mathrm{g}} &= 33 & \Longrightarrow \quad & y_{\mathrm{b}} = 33, \\
\text{across arc (b,c):} & & y_{\mathrm{c}} - y_{\mathrm{b}} &= 65 & \Longrightarrow \quad & y_{\mathrm{c}} = 98, \\
\text{across arc (f,b):} & & y_{\mathrm{b}} - y_{\mathrm{f}} &= 48 & \Longrightarrow \quad & y_{\mathrm{f}} = -15, \\
\text{across arc (f,a):} & & y_{\mathrm{a}} - y_{\mathrm{f}} &= 56 & \Longrightarrow \quad & y_{\mathrm{a}} = 41, \\
\text{across arc (a,d):} & & y_{\mathrm{d}} - y_{\mathrm{a}} &= 28 & \Longrightarrow \quad & y_{\mathrm{d}} = 69.
\end{aligned}$$

Now that we know the dual variables, the dual slacks for the arcs not in the spanning tree $\mathcal{T}$ can be computed using

$$z_{ij} = y_i + c_{ij} - y_j, \qquad (i,j) \notin \mathcal{T}$$

(which is just the dual feasibility condition solved for z_{ij}). These values are shown on the nontree arcs in Figure 14.6.

From duality theory, we know that the current tree solution is optimal if all the flows are nonnegative and if all the dual slacks are nonnegative. The tree solution shown in Figure 14.6 satisfies the first condition but not the second. That is, it is primal feasible but not dual feasible. Hence, we can apply the primal simplex method to move from this solution to an optimal one. We take up this task in the next section.

3. The Primal Network Simplex Method

Each of the variants of the simplex method presented in earlier chapters of this book can be applied to network flow problems. It would be overkill to describe them all here in the context of networks. However, they are all built on two simple algorithms: the primal simplex method (for problems that are primal feasible) and the dual simplex method (for problems that are dual feasible). We discuss them both in detail.

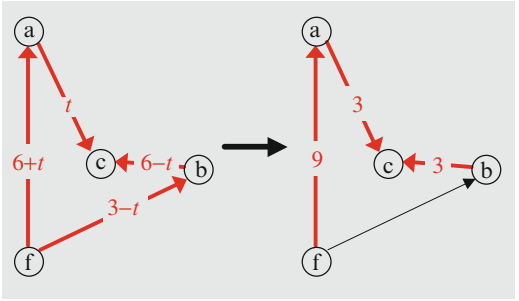


FIGURE 14.7. The cycle produced by including the entering arc with the spanning tree. As the flow t on the entering arc increases, eventually the flow on arc (f,b) becomes zero (when $t = 3$). Hence, arc (f,b) is the leaving arc.

We shall describe the primal network simplex method by continuing with our example. As mentioned above, the tree shown in Figure 14.6 is primal feasible but not dual feasible. The basic idea that defines the primal simplex method is to pick a nontree arc that is dual infeasible and let it enter the tree (i.e., become basic) and then readjust everything so that we still have a tree solution.

The First Iteration. For our first pivot, we let arc (a,c) enter the tree using a *primal pivot*. In a primal pivot, we add flow to the entering variable, keeping all other nontree flows set to zero and adjusting the tree flows appropriately to maintain flow balance. Given any spanning tree, adding an extra arc must create a cycle (why?). Hence, the current spanning tree together with the entering arc must contain a cycle. The flows on the cycle must change to accommodate the increasing flow on the entering arc. The flows on the other tree arcs remain unchanged. In our example, the cycle is: "a", "c", "b", "f". This cycle is shown in Figure 14.7 with flows adjusted to take into account a flow of t on the entering arc. As t increases, eventually the flow on arc (f,b) decreases to zero. Hence, arc (f,b) is the leaving arc. Updating the flows is easy; just take $t = 3$ and adjust the flows appropriately.

With a little thought, one realizes that the selection rule for the leaving arc in a primal pivot is as follows:

Leaving arc selection rule:

- The leaving arc must be oriented along the cycle in the reverse direction from the entering arc, and
- Among all such arcs, it must have the smallest flow.

Also, the flows on the cycle get updated as follows:

Primal flows update:

- Flows oriented in the same direction as the leaving arc are decreased by the amount of flow that was on the leaving arc whereas flows in the opposite direction are increased by this amount.

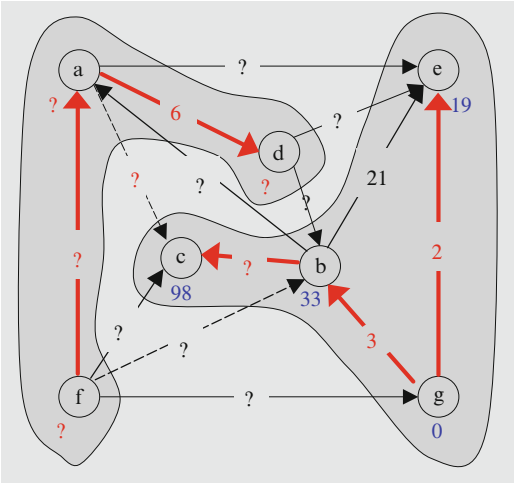


FIGURE 14.8. The two disjoint trees. Primal and dual values that remained unchanged are shown, whereas those that need to be updated are shown as *question marks*.

The next issue is how to update the dual variables. To this end, note that if we delete the leaving arc from the spanning tree (without concurrently adding the entering arc), we disconnect it into two disjoint trees. In our example, one tree contains nodes "a", "d" and "f" while the second tree contains the other nodes. Figure 14.8 shows the two disjoint trees. Recalling that the dual variables are calculated starting with the root node and working up the spanning tree, it is clear that the dual variables on the subtree containing the root node remain unchanged, whereas those on the other subtree must change. For the current pivot, the other subtree consists of nodes "a", "d", and "f". They all get incremented by the same fixed amount, since the only change is that the arc by which we bridged from the root-containing tree to this other tree has changed from the leaving arc to the entering arc. Looking at node "a" and using tildes to denote values after being changed, we see that

$$\begin{aligned}\tilde{y}_{\mathrm{a}} &= \tilde{y}_{\mathrm{c}} - c_{\mathrm{ac}} \\ &= y_{\mathrm{c}} - c_{\mathrm{ac}},\end{aligned}$$

whereas

$$z_{\mathrm{ac}} = y_{\mathrm{a}} + c_{\mathrm{ac}} - y_{\mathrm{c}}.$$

Combining these two equations, we get

$$\tilde{y}_{\mathrm{a}} = y_{\mathrm{a}} - z_{\mathrm{ac}}.$$

That is, the dual variable at node "a" gets decremented by $z_{\mathrm{ac}} = -9$. Of course, all of the dual variables on this subtree get decremented by this same amount. In general, the dual variable update rule can be stated as follows:

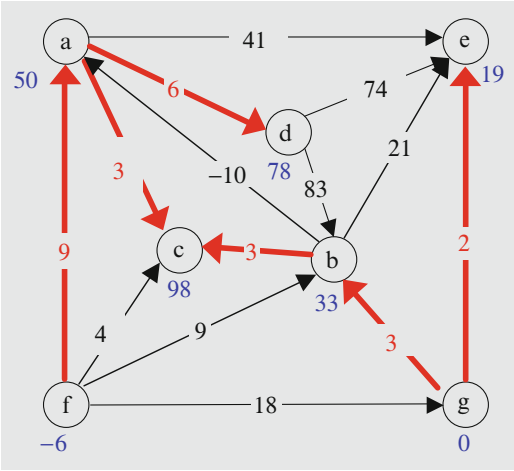


FIGURE 14.9. The tree solution at the end of the first iteration.

Dual variables update:

- If the entering arc crosses from the root-containing tree to the non-root-containing tree, then increase all dual variables on the non-root-containing tree by the dual slack of the entering arc.
- Otherwise, decrease these dual variables by this amount.

Finally, we must update the dual slacks. The only dual slacks that change are those that span across the two trees since, for these nodes, either the head or the tail dual variable changes, while the other does not. Those that span the two subtrees in the same direction as the entering arc must be decreased by z_{ac}, whereas those that bridge the two trees in the opposite direction must be increased by this amount. For our example, six nontree arcs, (f,g), (f,b), (f,c), (d,b), (d,e), and (a,e), span in the same direction as the entering arc. They all must be decreased by -9. That is, they must be increased by 9. For example, the dual slack on arc (f,c) changes from -5 to 4. Only one arc, (b,a), spans in the other direction. It must be decreased by 9. The updated solution is shown in Figure 14.9. The general rule for updating the dual slacks is as follows:

Dual slacks update:

- The dual slacks corresponding to those arcs that bridge in the same direction as the entering arc get decremented by the old dual slack on the entering arc, whereas those that correspond to arcs bridging in the opposite direction get incremented by this amount.

The Second Iteration. The tree solution shown in Figure 14.9 has only one remaining infeasibility: $z_{\text{ba}} = -10$. Arc (b,a) must therefore enter the spanning tree. Adding it, we create a cycle consisting of nodes “a”, “b”, and “c”. The leaving

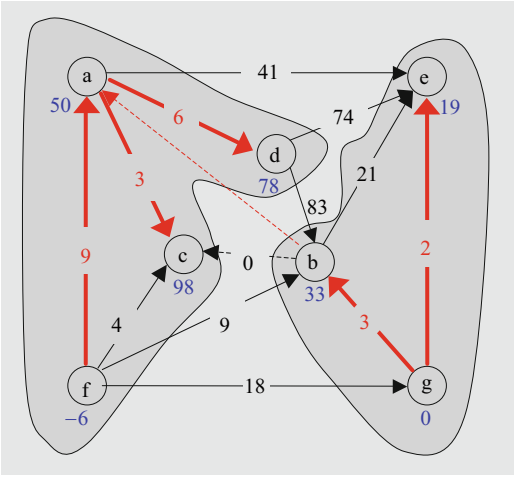


FIGURE 14.10. The two disjoint subtrees arising in the second iteration.

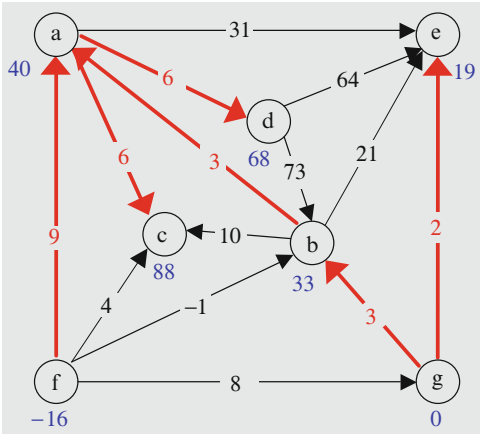


FIGURE 14.11. The tree solution at the end of the second iteration. To get from the spanning tree in Figure 14.9 to here, we let arc (b,a) enter and arc (b,c) leave.

arc must be pointing in the opposite direction from the entering arc. Here, there is only one such arc, (b,c). It must be the leaving arc. The leaving arc's flow decreases from 3 to 0. The flow on the other two cycle arcs must increase by 3 to preserve flow balance.

The two subtrees formed by removing the leaving arc are shown in Figure 14.10 The dual variables on the non-root-containing subtree get incremented by the dual slack on the entering arc $z_{ba} = -10$. The dual slacks for the spanning arcs also change by 10 either up or down depending on which way they bridge the two subtrees. The resulting tree solution is shown in Figure 14.11.

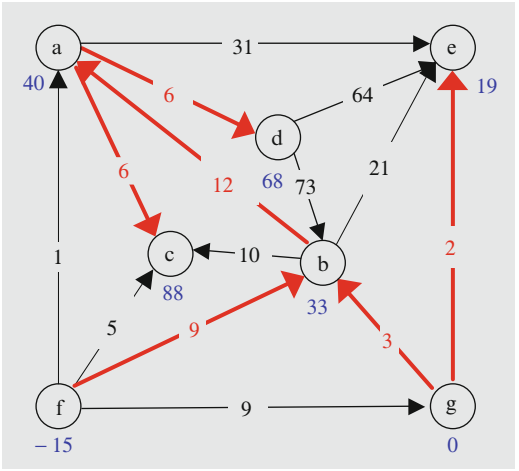


FIGURE 14.12. The tree solution at the end of the third iteration. To get from the spanning tree in Figure 14.11 to here, we let arc (f,b) enter and arc (f,a) leave. This tree solution is the *optimal* solution to the problem.

The Third and Final Iteration. The tree solution shown in Figure 14.11 has one infeasibility: $z_{\text{fb}} = -1$. Hence, arc (f,b) must enter the spanning tree. The leaving arc must be (f,a). Leaving the details of updating to the reader, the resulting tree solution is shown in Figure 14.12. It is both primal and dual feasible—hence optimal.

4. The Dual Network Simplex Method

In the previous section, we developed simple rules for the primal network simplex method, which is used in situations where the tree solution is primal feasible but not dual feasible. When a tree solution is dual feasible but not primal feasible, then the dual network simplex method can be used. We shall define this method now. Consider the tree solution shown in Figure 14.13. It is dual feasible but not primal feasible (since $x_{\text{db}} < 0$). The basic idea that defines the dual simplex method is to pick a tree arc that is primal infeasible and let it leave the spanning tree (i.e., become nonbasic) and then readjust everything to preserve dual feasibility.

The First Iteration. For the first iteration, we need to let arc (d,b) leave the spanning tree using a *dual pivot*, which is defined as follows. Removing arc (d,b) disconnects the spanning tree into two disjoint subtrees. The entering arc must be one of the arcs that spans across the two subtrees so that it can reconnect them into a spanning tree. That is, it must be one of

$$\text{(a,e)}, \quad \text{(a,d)}, \quad \text{(b,e)}, \quad \text{or} \quad \text{(g,e)}.$$

See Figure 14.14. To see how to decide which it must be, we need to consider carefully the impact of each possible choice.

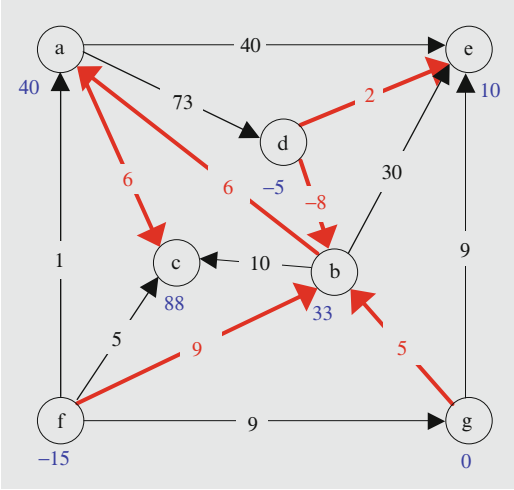


FIGURE 14.13. A tree solution that is dual feasible but not primal feasible.

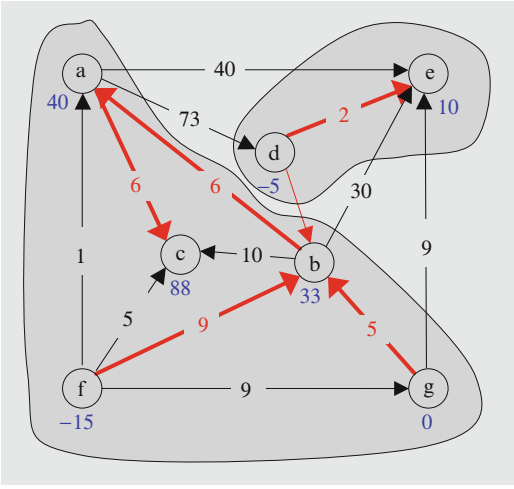


FIGURE 14.14. The two subtrees for the first pivot of the dual simplex method.

To this end, let us consider the general situation. As mentioned above, the spanning tree with the leaving arc removed consists of two disjoint trees. The entering arc must reconnect these two trees.

First, consider a reconnecting arc that connects in the same direction as the leaving arc. When we add flow to this prospective entering arc, we will have to decrease flow on the leaving arc to maintain flow balance. Therefore, the leaving

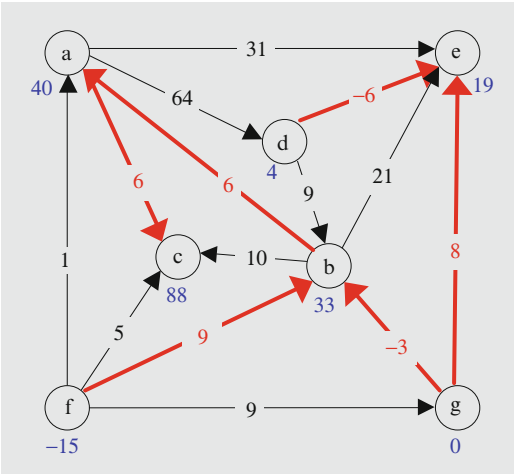


FIGURE 14.15. The tree solution after the first pivot.

arc's flow, which is currently negative, can't be raised to zero. That is, the leaving arc can't leave. This is no good.

Now suppose that the reconnecting arc connects in the opposite direction. If it were to be the entering arc, then its dual slack would drop to zero. All other reconnecting arcs pointing in the same direction would drop by the same amount. To maintain nonnegativity of all the others, we must pick the one that drops the least. We can summarize the rule as follows:

Entering arc selection rule:

- The entering arc must bridge the two subtrees in the opposite direction from the leaving arc, and
- Among all such arcs, it must have the smallest dual slack.

In our example, all bridging arcs point in the opposite direction from the leaving arc. The one with the smallest dual slack is (g,e) whose slack is $z_{ge} = 9$. This arc must be the entering arc.

We have now determined both the entering and leaving arcs. Hence, the new spanning tree is determined and therefore, in principle, all the variables associated with this new spanning tree can be computed. Furthermore, the rules for determining the new values by updating from the previous ones are the same as in the primal network simplex method. The resulting tree solution is shown in Figure 14.15.

The Second Iteration. For the second pivot, there are two choices for the leaving arc: (g,b) and (d,e). Using the most infeasible, we choose (d,e). We remove this arc from the spanning tree to produce two subtrees. One of the subtrees consists of just the node "d" all by itself while the other subtree consists of the rest of the nodes. Remembering that the reconnecting arc must bridge the two subtrees in the opposite direction, the only choice is (a,d). So this arc is the entering arc. Making the pivot, we arrive at the optimal tree solution shown in Figure 14.12.

5. Putting It All Together

As we saw in Chap. 5, for linear programming the primal and the dual simplex methods form the foundation on which one can build a few different variants of the simplex method. The same is true here in the context of network flows.

For example, one can build a two-phased procedure in which one first uses the dual network simplex method (with costs artificially and temporarily altered to ensure dual feasibility of an initial tree solution) to find a primal feasible solution and then uses the primal network simplex method to move from the feasible solution to an optimal one.

Alternatively, one can use the primal network simplex method (with supplies temporarily altered to ensure primal feasibility of an initial tree solution) to find a dual feasible solution and then use the dual network simplex method (with the original supplies) to move from the dual feasible solution to an optimal one.

Finally, as described for linear programming in Chap. 7, one can define a parametric self-dual method in which primal pivots and dual pivots are intermingled as needed so as to reduce a perturbation parameter μ from ∞ to zero.

Since there is nothing new in how one builds the network versions of these algorithms from the basic primal and dual simplex pivots, we don't go through any examples here. Instead, we just mention one final observation about the dual variables, the y_i's. Namely, they are not needed anywhere in the performance of a primal or a dual pivot. Hence, their calculation is entirely optional and can be skipped altogether or simply deferred to the end.

For completeness, we end this section by giving a step-by-step description of the *self-dual network simplex method*. The steps are as follows:

(1) *Identify a spanning tree*—any one will do (see Exercise 14.14). Also identify a root node.

(2) Compute *initial primal flows* on the tree arcs by assuming that nontree arcs have zero flow and the total flow at each node must be balanced. For this calculation, the computed primal flows may be negative. In this case, the initial primal solution is not feasible. The calculation is performed working from leaf nodes inward.

(3) Compute *initial dual values* by working out from the root node along tree arcs using the formula

$$y_j - y_i = c_{ij},$$

which is valid on tree arcs, since the dual slacks vanish on these arcs.

(4) Compute *initial dual slacks* on each nontree arc using the formula

$$z_{ij} = y_i + c_{ij} - y_j.$$

Again, some of the z_{ij}'s might be nonnegative. This is okay (for now), but it is important that they satisfy the above equality.

(5) *Perturb* each primal flow and each dual slack that has a negative initial value by adding a positive scalar μ to each such value.

(6) *Identify a range* $\mu_{\text{MIN}} \leq \mu \leq \mu_{\text{MAX}}$ over which the current solution is optimal (on the first iteration, μ_{MAX} will be infinite).

(7) *Check the stopping rule:* if $\mu_{\text{MIN}} \leq 0$, then set $\mu = 0$ to recover an optimal solution. While not optimal, perform each of the remaining steps and then return to recheck this condition.

(8) *Select an arc* associated with the inequality $\mu_{\text{MIN}} \leq \mu$ (if there are several, pick one arbitrarily). If this arc is a nontree arc, then the current pivot is a *primal pivot*. If, on the other hand, it is a tree arc, then the pivot is a *dual pivot*.
   (a) If the pivot is a primal pivot, the arc identified above is the *entering arc*. Identify the associated *leaving arc* as follows. First, add the entering arc to the tree. With this arc added, there must be a cycle consisting of the entering arc and other tree arcs. The leaving arc is chosen from those arcs on the cycle that go in the opposite direction from the entering arc and having the smallest flow among all such arcs (evaluated at $\mu = \mu_{\text{MIN}}$).
   (b) If the pivot is a dual pivot, the arc identified above is the *leaving arc*. Identify the associated *entering arc* as follows. First, delete the leaving arc from the tree. This deletion splits the tree into two subtrees. The entering arc must bridge these two trees in the opposite direction to the leaving arc, and, among such arcs, it must be the one with the smallest dual slack (evaluated at $\mu = \mu_{\text{MIN}}$).

(9) *Update primal flows* as follows. Add the entering arc to the tree. This addition creates a cycle containing both the entering and leaving arcs. Adjust the flow on the leaving arc to zero, and then adjust the flows on each of the other cycle arcs as necessary to maintain flow balance.

(10) *Update dual variables* as follows. Delete the leaving arc from the old tree. This deletion splits the old tree into two subtrees. Let $\mathcal{T}_u$ denote the subtree containing the tail of the entering arc, and let $\mathcal{T}_v$ denote the subtree containing its head. The dual variables for nodes in $\mathcal{T}_u$ remain unchanged, but the dual variables for nodes in $\mathcal{T}_v$ get incremented by the old dual slack on the entering arc.

(11) *Update dual slacks* as follows. All dual slacks remain unchanged except for those associated with nontree arcs that bridge the two subtrees $\mathcal{T}_u$ and $\mathcal{T}_v$. The dual slacks corresponding to those arcs that bridge in the same direction as the entering arc get decremented by the old dual slack on the entering arc, whereas those that correspond to arcs bridging in the opposite direction get incremented by this amount.

As was said before and should now be clear, there is no need to update the dual variables from one iteration to the next; that is, step 10 can be skipped.

6. The Integrality Theorem

In this section, we consider network flow problems for which all the supplies and demands are integers. Such problems are called *network flow problems with integer data*. As we explained in Sect. 14.2, for network flow problems, basic primal

solutions are computed without any multiplication or division. The following important theorem follows immediately from this property:

THEOREM 14.2. *Integrality Theorem. For network flow problems with integer data, every basic feasible solution and, in particular, every basic optimal solution assigns integer flow to every arc.*

This theorem is important because many real-world network flow problems have integral supplies/demands and require their solutions to be integral too. This integrality restriction typically occurs when one is shipping indivisible units through a network. For example, it wouldn't make sense to ship one third of a car from an automobile assembly plant to one dealership with the other two thirds going to another dealership.

Problems that are linear programming problems with the additional stipulation that the optimal solution values must be integers are called *integer programming problems*. Generally speaking, these problems are much harder to solve than linear programming problems (see Chap. 23). However, if the problem is a network flow problem with integer data, it can be solved efficiently using the simplex method to compute a basic optimal solution, which the integrality theorem tells us will be integer valued.

6.1. König's Theorem. In addition to its importance in real-world optimization problems, the integrality theorem also has many applications to the branch of mathematics called combinatorics. We illustrate with just one example.

THEOREM 14.3. *König's Theorem. Suppose that there are n girls and n boys, that every girl knows exactly k boys, and that every boy knows exactly k girls. Then n marriages can be arranged with everybody knowing his or her spouse.*

Before proving this theorem it is important to clarify its statement by saying that the property of "knowing" is symmetric (for example, knowing in the biblical sense). That is, if a certain girl knows a certain boy, then this boy also knows this girl.

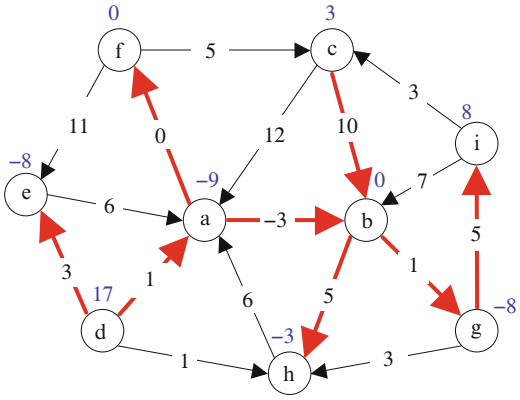
PROOF. Consider a network with nodes $g_1, g_2, \ldots, g_n, b_1, b_2, \ldots, b_n$ and an arc from g_i to b_j if girl i and boy j know each other. Assign one unit of supply to each girl node and a unit of demand to each boy node. Assign arbitrary objective coefficients to create a well-defined network flow problem. The problem is guaranteed to be feasible: just put a flow of $1/k$ on each arc (the polygamists in the group might prefer this nonintegral solution). By the integrality theorem, the problem has an integer-valued solution. Clearly, the flow on each arc must be either zero or one. Also, each girl node is the tail of exactly one arc having a flow of one. This arc points to her intended mate. □

Exercises

In solving the following problems, the network pivot tool can be used to check your arithmetic:

www.princeton.edu/∼rvdb/JAVA/network/nettool/netsimp.html

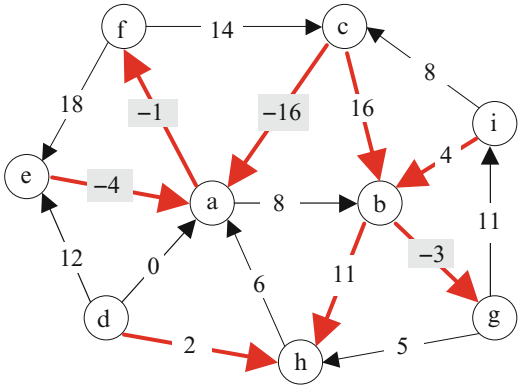
14.1 Consider the following network flow problem:



Numbers shown above the nodes are supplies (negative values represent demands) and numbers shown above the arcs are unit shipping costs. The darkened arcs form a spanning tree.

(a) Compute primal flows for each tree arc.
(b) Compute dual variables for each node.
(c) Compute dual slacks for each nontree arc.

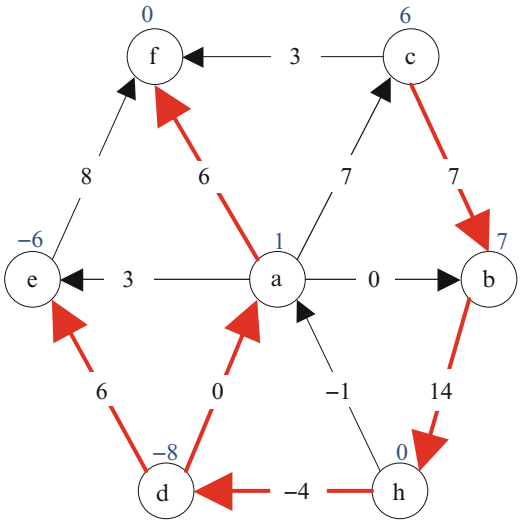
14.2 Consider the tree solution for the following minimum cost network flow problem:



The numbers on the tree arcs represent primal flows while numbers on the nontree arcs are dual slacks.

(a) Using the largest–coefficient rule in the dual network simplex method, what is the leaving arc?
(b) What is the entering arc?
(c) After *one* pivot, what is the new tree solution?

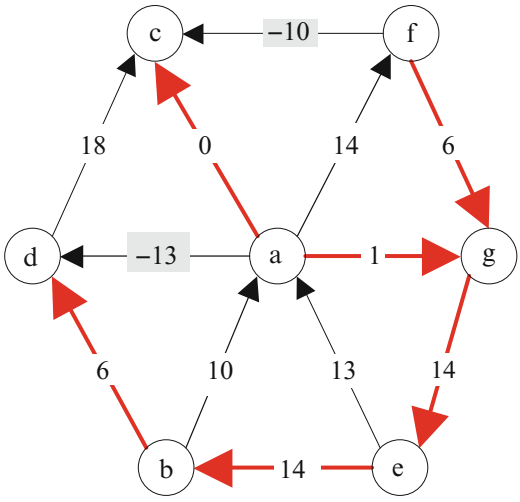
14.3 Consider the following network flow problem:



The numbers above the nodes are supplies (negative values represent demands) and numbers shown above the arcs are unit shipping costs. The darkened arcs form a spanning tree.

(a) Compute primal flows for each tree arc.
(b) Compute dual variables for each node.
(c) Compute dual slacks for each nontree arc.

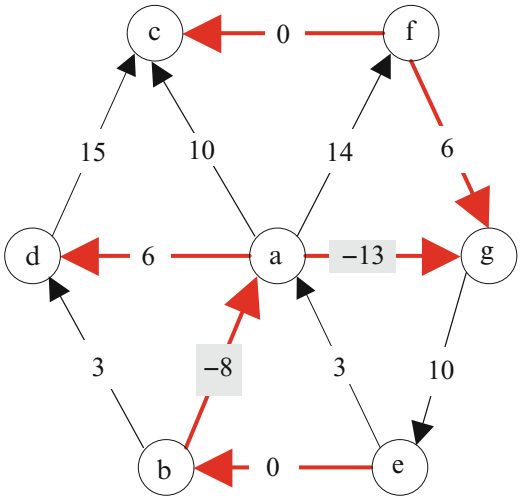
14.4 Consider the tree solution for the following minimum cost network flow problem:



The numbers on the tree arcs represent primal flows while numbers on the nontree arcs are dual slacks.

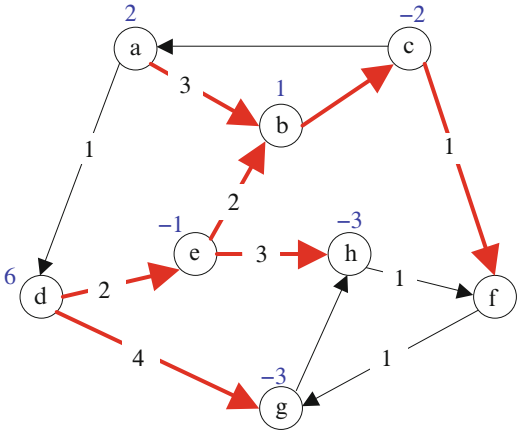
(a) Using the largest–coefficient rule in the primal network simplex method, what is the entering arc?
(b) What is the leaving arc?
(c) After *one* pivot, what is the new tree solution?

14.5 Consider the tree solution for the following minimum cost network flow problem:



The numbers on the tree arcs represent primal flows while numbers on the nontree arcs are dual slacks.
(a) Using the largest–coefficient rule in the dual network simplex method, what is the leaving arc?
(b) What is the entering arc?
(c) After *one* pivot, what is the new tree solution?

14.6 Solve the following network flow problem starting with the spanning tree shown.



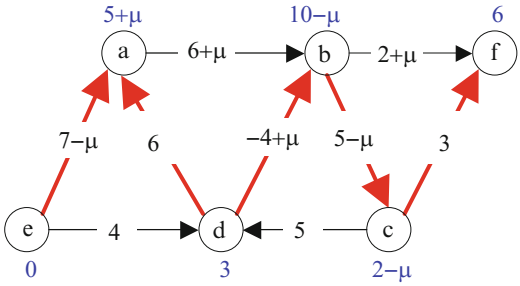
The numbers displayed next to nodes are supplies(+)/demands(−). Numbers on arcs are costs. Missing data should be assumed to be zero. The bold arcs represent an initial spanning tree.

14.7 Solve Exercise 2.11 using the self-dual network simplex method.

14.8 Using today's date (MMYY) for the seed value, solve ten problems using the network simplex pivot tool:

www.princeton.edu/∼rvdb/JAVA/network/challenge/netsimp.html

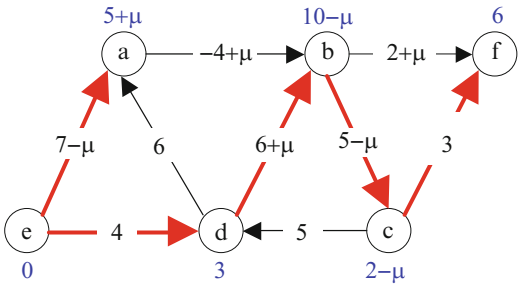
14.9 Consider the following tree solution for a minimum cost network flow problem:



As usual, bold arcs represent arcs on the spanning tree, numbers next to the bold arcs are primal flows, numbers next to non-bold arcs are dual slacks, and numbers next to nodes are dual variables.

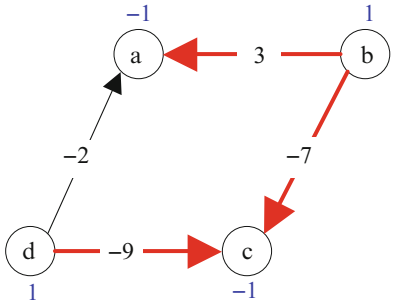
(a) For what values of μ is this tree solution optimal?
(b) What are the entering and leaving arcs?
(c) After *one* pivot, what is the new tree solution?
(d) For what values of μ is the new tree solution optimal?

14.10 Consider the following tree solution for a minimum cost network flow problem:



(a) For what values of μ is this tree solution optimal?
(b) What are the entering and leaving arcs?
(c) After *one* pivot, what is the new tree solution?
(d) For what values of μ is the new tree solution optimal?

14.11 Consider the following minimum cost network flow problem



As usual, the numbers on the arcs represent the flow costs and numbers at the nodes represent supplies (demands are shown as negative supplies). The arcs shown in bold represent a spanning tree. If the solution corresponding to this spanning tree is optimal prove it, otherwise find an optimal solution using this tree as the initial spanning tree.

14.12 Suppose that a square submatrix of $\tilde{A}$ is invertible. Show that the arcs corresponding to the columns of this submatrix form a spanning tree.

14.13 Show that a spanning tree on m nodes must have exactly $m - 1$ arcs.

14.14 Define an algorithm that takes as input a network and either finds a spanning tree or proves that the network is not connected.

14.15 Give an example of a minimum-cost network flow problem with all arc costs positive and the following counterintuitive property: if the supply at a particular source node and the demand at a particular sink node are simultaneously reduced by one unit, then the optimal cost increases.

14.16 Consider a possibly disconnected network $(\mathcal{N}, \mathcal{A})$. Two nodes i and j in $\mathcal{N}$ are said to be *connected* if there is a path from i to j (recall that paths can traverse arcs backwards or forwards). We write $i \sim j$ if i and j are connected.

(a) Show that "$\sim$" defines an *equivalence relation*. That is, it has the following three properties:
   (i) (Reflexivity) for all $i \in \mathcal{N}$, $i \sim i$;
   (ii) (Symmetry) for all $i, j \in \mathcal{N}$, $i \sim j$ implies that $j \sim i$;
   (iii) (Transitivity) for all $i, j, k \in \mathcal{N}$, $i \sim j$ and $j \sim k$ implies that $i \sim k$.

Using the equivalence relation, we can partition $\mathcal{N}$ into a collection of subsets of *equivalence classes* $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ such that two nodes are connected if and only if they belong to the same subset. The number k is called the number of *connected components*.

(b) Show that the rank of the node–arc incidence matrix A is exactly $m - k$ (recall that m is the number of rows of A).

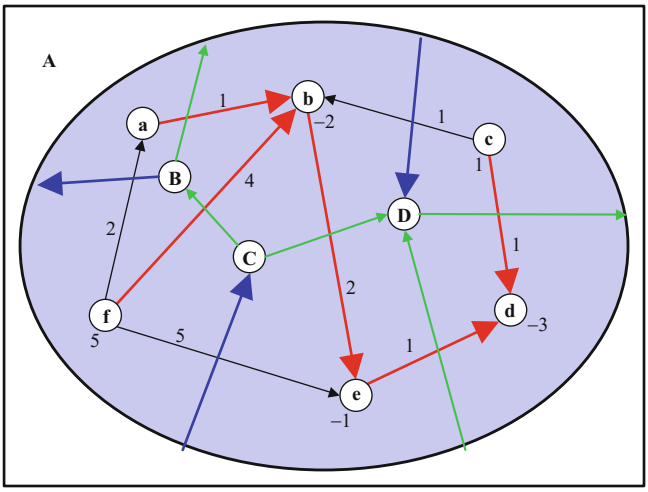


FIGURE 14.16. The primal network has nodes "a" through "f". The corresponding dual network has nodes "A" through "D" (node "A" is "at infinity"). A primal spanning tree is shown. It consists of five arcs: (a,b), (f,b), (b,e), (e,d), and (c,d). The corresponding dual spanning tree consists of three arcs: (B,A), (A,C), and (D,A). Primal costs are shown along the primal arcs and supplies/demands are shown at the primal nodes.

14.17 One may assume without loss of generality that every node in a minimum cost network flow problem has at least two arcs associated with it. Why?

14.18 The sum of the dual slacks around any cycle is a constant. What is that constant?

14.19 *Planar Networks*. A network is called *planar* if the nodes and arcs can be laid out on the two-dimensional plane in such a manner that no two arcs cross each other (it is allowed to draw the arcs as curves if necessary). All of the networks encountered so far in this chapter have been planar. Associated with each planar network is a geometrically defined dual network. The purpose of this problem is to establish the following interesting fact:

A dual network simplex pivot is precisely a primal network simplex method applied to the dual network.

Viewed geometrically, the nodes of a planar graph are called *vertices* and the arcs are called *edges*. Consider a specific connected planar network. If one were to delete the vertices and the edges from the plane, one would be left with a disjoint collection of subsets of the plane. These subsets are called *faces*. Note that there is one unbounded face. It is a face just like the other bounded ones. An example of a connected planar network with its faces labeled A through D is shown in Figure 14.16.

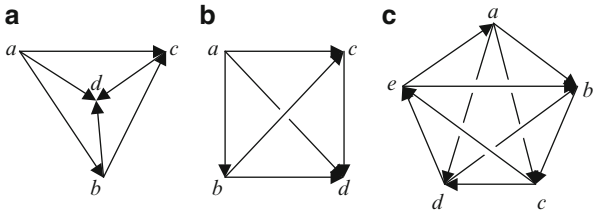
Dual nodes. Associated with each connected planar network is a *dual network* defined by interchanging vertices and faces. That is, place a dual vertex in the center of each primal face. Note: the dual vertex corresponding to the unbounded primal face could be placed anywhere in the unbounded face but we choose to put it *at infinity*. In this way, dual edges (defined next) that have a head or a tail at this node can run off to infinity in any direction.

Dual arcs. Connect with a dual edge any pair of dual nodes whose corresponding primal faces share an edge. Each dual edge crosses exactly one primal edge. The directionality of the dual edge is determined as follows: first, place a vector along the corresponding primal edge pointing in the direction of the primal arc, and then rotate it counterclockwise until it is tangent to the dual edge. The vector now defines the direction for the dual arc.

Dual spanning tree. Consider a spanning tree on the primal network and suppose that a primal–dual tree solution is given. We define a *spanning tree* on the dual network as follows. A dual edge is on the dual network's spanning tree if and only if the corresponding primal edge is not on the primal network's spanning tree.

Dual flows and dual dual-slacks. The numerical arc data for the dual network is inherited directly from the primal. That is, flows on the dual tree arcs are exactly equal to the dual slacks on the associated primal nontree arcs. And, the dual slacks on the dual nontree arcs are exactly equal to the primal flows on the associated primal tree arcs. Having specified numerical data on the arcs of the dual network, it is fairly straightforward to determine values for supplies/demands at the nodes and shipping costs along the arcs that are consistent with these numerical values.

(a) Which of the following networks are planar:



(b) A network is called *complete* if there is an arc between every pair of nodes. If a complete network with m nodes is planar, then every network with m nodes is planar. Prove it.

(c) Show that a nonplanar network must have five or more nodes.

(d) As always, let m denote the number of nodes and let n denote the number of arcs in a network. Let f denote the number of faces in a planar network. Show by induction on f that $m = n - f + 2$.

(e) Show that the dual spanning tree defined above is in fact a spanning tree.
(f) Show that a dual pivot for a minimum cost network flow problem defined on the primal network is precisely the same as a primal pivot for the corresponding network flow problem on the dual network.
(g) Using the cost and supply/demand information given for the primal problem in Figure 14.16, write down the primal problem as a linear programming problem.
(h) Write down the dual linear programming problem that one derives algebraically from the primal linear programming problem.
(i) Using the spanning tree shown in Figure 14.16, compute the primal flows, dual variables, and dual slacks for the network flow problem associated with the primal network.
(j) Write down the flow and slacks for the network flow problem associated with the dual network.
(k) Find arc costs and node supplies/demands for the dual network that are consistent with the flows and slacks just computed.
(l) Write down the linear programming problem associated with the network flow problem on the dual network.

Notes

The classical reference is Ford and Fulkerson (1962). More recent works include the books by Christofides (1975), Lawler (1976), Bazaraa et al. (1977), Kennington and Helgason (1980), Jensen and Barnes (1980), Bertsekas (1991), and Ahuja et al. (1993).

The two "original" algorithms for solving minimum-cost network flow problems are the *network simplex method* developed by Dantzig (1951a) and the *primal–dual method* developed by Ford and Fulkerson (1958). The self-dual algorithm described in this chapter is neither of these. In fact, it resembles the "out-of-kilter" method described by Ford and Fulkerson (1962).