Chapter 13

Optimization Problems for Graphs and Networks

In this chapter we present procedures for solving a number of combinatorial optimization problems. We have encountered such problems throughout the book and discussed some in detail in Sections 12.4–12.6. The problems discussed will all be translated into problems involving graphs or digraphs. More generally, they will be translated into problems involving graphs or digraphs in which each edge or arc has one or more *nonnegative* real numbers assigned to it. Such a graph or digraph is called a *network* or a *directed network*, respectively.

It should be mentioned that we have chosen problems to discuss for which good solutions (good algorithms) are known. Not all combinatorial optimization problems have good solutions. The traveling salesman problem, discussed in Sections 2.4 and 11.5, is a case in point. In such a case, as we pointed out in Section 2.18, we seek good algorithms that solve the problem in special cases or that give approximate solutions. Discussion of these approaches is beyond the scope of this book.

13.1 MINIMUM SPANNING TREES

13.1.1 Kruskal's Algorithm

In Examples 3.28–3.32 we described the problem of finding a minimum spanning tree of a graph G with weights, that is, a network. This is a spanning subgraph of G that forms a tree and that has the property that no other spanning tree has a smaller sum of weights on its edges. Here we discuss a simple algorithm for finding a minimum spanning tree of a network G. The algorithm will either find a minimum spanning tree or conclude that the network does not have a spanning tree. By Theorem 3.19, the latter implies that the network is disconnected. The problem of finding a minimum spanning tree has a wide variety of applications: for example, in the planning of large-scale transportation, communication, or distribution networks;

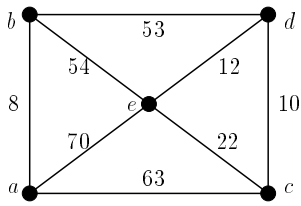


Figure 13.1: A network of road distances between villages.

in reducing data storage; and in data mining. We mentioned some of these in Section 3.5.4. See Ahuja, Magnanti, and Orlin [1993] or Graham and Hell [1985] for a survey and references on many applications.

According to Graham and Hell [1985], the earliest algorithms for finding a minimum spanning tree apparently go back to Borůvka [1926a,b][1], who was interested in the most economical layout for a power-line network. Earlier work on this problem is due to the anthropologist Czekanowski [1909, 1911, 1928] in his work on classification schemes.

Let us recall the problem by continuing with Example 3.29. Suppose that there are five villages in a remote region and that the road distances between them are as shown in the network of Figure 13.1. We wish to put in telephone lines along the existing roads in such a way that every pair of villages is linked by telephone service and we minimize the total number of miles of telephone line. The problem amounts to finding a minimum spanning tree of the network of Figure 13.1.

In finding a spanning tree of such a network that minimizes the sum of the weights of its edges, we will build up the tree one edge at a time, and we will be *greedy*; that is, we will add edges of smallest weight first. It turns out that being greedy works. Specifically, let us order the edges of the network G in order of increasing weight. In our example, we order them as follows: $\{a,b\}$, $\{c,d\}$, $\{d,e\}$, $\{c,e\}$, $\{b,d\}$, $\{b,e\}$, $\{a,c\}$, $\{a,e\}$. In case of ties, the ordering is arbitrary. We examine the edges in this order, deciding for each edge whether or not to include it in the spanning tree T. It is included as long as it does not form a circuit with some already included edges. This algorithm is called Kruskal's Algorithm (it is due to Kruskal [1956]) and can be summarized as follows.

Algorithm 13.1: Kruskal's Minimum Spanning Tree Algorithm

Input: A network G on $n > 1$ vertices.

Output: A set of edges defining a minimum spanning tree of G or the message that G is disconnected.

Step 1. Arrange the edges of G by increasing order of weight. (Order arbitrarily in case of ties.) Set $T = \emptyset$. (T is the set of edges of the minimum spanning tree.)

Step 2. If every edge of G has been examined, stop and output the message that G is disconnected. Otherwise, examine the first unexamined edge in the ordered

[1] Nešetřil, Milková, and Nešetřilová [2001] is an English translation of Borůvka's works in Czech on minimum spanning trees, with commentaries and historical development.

list and include it in the set T if and only if it does not form a circuit with some edges already in T. If the edge is added to T, go to Step 3. Otherwise, repeat Step 2.

Step 3. If T has $n-1$ edges, stop and output T. Otherwise, go to Step 2.

Theorem 13.1 If G is a connected network on n vertices, Kruskal's Algorithm will terminate with a minimum spanning tree T of $n-1$ edges. If G is a disconnected network, the algorithm will terminate with the message that G is disconnected after having examined all the edges and not yet having put $n-1$ edges into T.

Before presenting a proof of Theorem 13.1, we illustrate the algorithm on our example of Figure 13.1. We start by including edge $\{a, b\}$ in T. Now the number of vertices of G is $n = 5$, and T does not yet have $n - 1 = 4$ edges, so we return to Step 2. In Step 2 we note that not every edge of G has been examined. We examine edge $\{c, d\}$, adding it to T because it does not form a circuit with edges in T. Now T has two edges, so we return to Step 2. Since the first unexamined edge $\{d, e\}$ does not form a circuit with existing edges in T, we add this edge to T. Now T has three edges, so we return to Step 2. The next edge examined, $\{c, e\}$, does form a circuit with edges $\{c, d\}$ and $\{d, e\}$ of T, so we do not add it to T. We repeat Step 2 for the next edge, $\{b, d\}$. This is added to T. Since T now has four edges in it, we stop and conclude that these edges define a minimum spanning tree of G.

Let us note that Kruskal's Algorithm can be applied to a graph G without weights if we simply list the edges in arbitrary order. Then the algorithm will produce a spanning tree of G or it will conclude that G is disconnected.

Let us consider the computational complexity of Kruskal's Algorithm. Disregarding the step of putting the edges in order, the algorithm takes at most e steps, one for each edge. The reader should not confuse "step" here with "step" in Algorithm 13.1. Some edges require both Steps 2 and 3 of the algorithm. So, in some sense, up to $2e$ steps are required. However, $O(e) = O(2e)$ (see Section 2.18.1), so e steps and $2e$ steps are considered "the same."

How many steps does it take to order the edges in terms of increasing weight? This is simply a sorting problem, of the kind discussed in Section 3.6.4. In that section we observed that sorting a set of e items can be performed in $ke \log_2 e$ steps, where k is a constant. Thus, the entire Kruskal's Algorithm can be performed in at most $e + ke \log_2 e \leq e + ke^2$ steps. Note that $e + ke^2 \leq n^2 + kn^4$, since $e \leq \binom{n}{2} \leq n^2$. Thus, the entire Kruskal's Algorithm can be performed in a number of steps that is a polynomial in either e or n. (In the terminology of Section 2.18, our crude discussion shows that we have an $O(e \log_2 e)$ algorithm. The best implementation of Kruskal's Algorithm, by Tarjan [1984a], finds a minimum spanning tree in $O(e\alpha(n, e))$ steps for a function $\alpha(n, e)$ that grows so slowly that for all practical purposes it can be considered a constant less than 6. See Ahuja, Magnanti, and Orlin [1993], Cheriton and Tarjan [1976], Graham and Hell [1985], or Tarjan [1983, 1984b].)

Example 13.1 Clustering and Data Mining (Ahuja, Magnanti, and Orlin [1993], Gower and Ross [1969], Zahn [1971]) In many practical problems

of detection, decisionmaking, or pattern recognition, especially in "data mining" of today's massive datasets in a wide variety of fields, we often seek to partition the data into natural groups or *clusters*. Clustering problems arise in applications involving molecular databases, astrophysical studies, geographic information systems, software measurement, worldwide ecological monitoring, medicine, and analysis of telecommunications and credit card data for fraud. (See Arratia and Lander [1990], Baritchi, Cook, and Holder [2000], Church [1997], Godehart [1990], Karger [1997], or Neil [1992].) The idea is that elements in the same cluster should be closely related, whereas elements in different clusters should not be as closely related, although how to define this precisely is subject to a variety of attempts to make it precise. (See, for example, Guénoche, Hansen, and Jaumard [1991], Hansen, Frank, and Jaumard [1989], Hansen and Jaumard [1987], Hansen, Jaumard, and Mladenovic [1998], or Mirkin [1996].) In many applications, the objects to be clustered are represented as points in two-dimensional Euclidean space (more generally, k-dimensional space). Here, Kruskal's Algorithm is widely used for clustering. At each iteration, the algorithm produces a set of edges that partition the graph into a set of trees. Each tree can be considered a cluster. The algorithm produces n partitions if there are n vertices, starting with the partition where each cluster has one vertex and ending with the partition where there is only one cluster consisting of all the points. Which of these partitions is "best" or "most useful" depends on defining an "objective function" precisely. ■

13.1.2 Proof of Theorem 13.1[2]

To prove Theorem 13.1, note that at each stage of the algorithm the edges in T define a graph that has no circuits. Suppose that the algorithm terminates with T having $n-1$ edges. Then, by Theorem 3.20, T is a tree, and hence a spanning tree of G. Thus, G is connected. Hence, if G is not connected, the algorithm will terminate without finding a set T with $n-1$ edges.

If G is connected, we shall show that the algorithm gives rise to a minimum spanning tree. First, we show that the algorithm does give rise to a spanning tree. Then we show that this spanning tree is minimum. The first observation follows trivially from Theorem 3.16 if the algorithm gives us a T of $n-1$ edges. Suppose that the algorithm terminates without giving us a T of $n-1$ edges. At the time of termination, let us consider the edges in T. These edges define a spanning subgraph of G, that we also refer to as T. This subgraph is not connected, for any connected spanning subgraph of G, having a spanning tree in it, must have at least $n-1$ edges. Hence, the subgraph T has at least two components. But since G is connected, there is in G an edge $\{x, y\}$ joining vertices in different components of T. Now $\{x, y\}$ cannot form a circuit with edges of T. Hence, when the algorithm came to examine this edge, it should have included it in T. Thus, this situation arises only if the algorithm was applied incorrectly.

[2] This subsection may be omitted.

We now know that the algorithm gives us a spanning tree T. Let S be a minimum spanning tree of G. If $S = T$, we are done. Suppose that $S \neq T$. Since $S \neq T$ and since both have the same number of edges $(n-1)$, there must be an edge in T that is not in S. In the order of edges by increasing weight, find the first edge $e_1 = \{x, y\}$ that is in T but not in S.

Since S is a spanning tree of G, there is a simple chain $C(x, y)$ from x to y in S. Now adding edge e_1 to S gives us a circuit. Thus, since T has no circuits, there is an edge e_2 on this circuit and hence on the chain $C(x, y)$ that is not in T. Let S' be the set of edges obtained from S by removing e_2 and adding e_1. Then S' defines a graph that is connected (why?) and has n vertices and $n-1$ edges, so by Theorem 3.16, S' is a spanning tree. We consider two cases.

Case 1: Edge e_2 precedes edge e_1 in the order of edges by increasing weights. In this case, e_2 was not put in T, so it must form a circuit D with edges of T that were examined before e_2. But e_1 is the first edge of T not in S, so all edges in D must be in S since they are in T. Thus, e_2 could not have been put in S, which is a contradiction. Case 1 is impossible.

Case 2: Edge e_1 precedes edge e_2. In this case, e_2 has weight at least as high as the weight of e_1. Thus, S' has its sum of weights at most the sum of weights of S, so S' is a minimum spanning tree since S is. Moreover, S' has one more edge in common with T than does S.

If $T \neq S'$, we repeat the argument for T and S', obtaining a minimum spanning tree S'' with one more edge in common with T than S'. Eventually, we find a minimum spanning tree that is the same as T. This completes the proof.

13.1.3 Prim's Algorithm

There is an alternative algorithm for finding a minimum spanning tree that is also a greedy algorithm. We present this algorithm here.

Algorithm 13.2: Prim's Minimum Spanning Tree Algorithm[3]

Input: A network G on $n > 1$ vertices.
Output: A minimum spanning tree of G or the message that G is disconnected.

Step 1. Set $T = \emptyset$. Pick an arbitrary vertex of G and put it in the tree T.

Step 2. Add to T that edge joining a vertex of T to a vertex not in T that has the smallest weight among all such edges. Pick arbitrarily in case of ties. If it is impossible to add any edge to T, stop and output the message that G is disconnected.

Step 3. If T has $n-1$ edges, stop and output T. Otherwise, repeat Step 2.

To illustrate this algorithm, let us again consider the graph of Figure 13.1 and start with vertex a. Then we add edge $\{a, b\}$ to T because it has the smallest weight

[3]This algorithm was discovered by Prim [1957] and is usually attributed to him. In fact, it seems to have been discovered previously by Jarník [1930] (Graham and Hell [1985]).

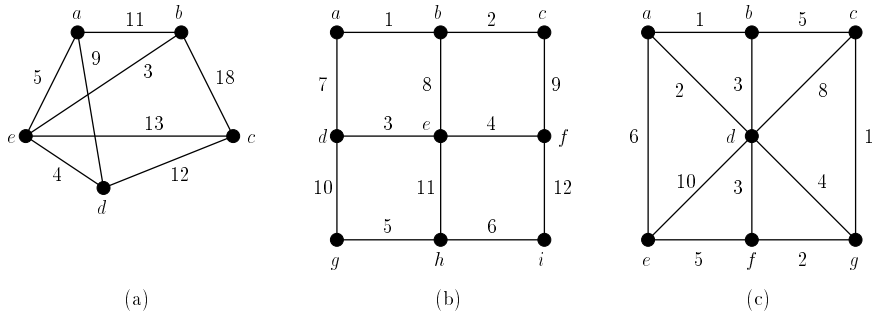


Figure 13.2: Networks for exercises of Section 13.1.

of the edges joining a. Next, we examine edges joining vertices a or b to vertices not in T: namely, c, d, or e. Edge $\{b, d\}$ has the smallest weight of all these edges; we add it to T. Next, we examine edges joining vertices a, b, or d to vertices not in T, namely c or e. Edge $\{d, c\}$ has the smallest weight of all of these edges. We add this edge to T. Finally, we examine edges joining a, b, c, or d to the remaining vertex not in T, namely e. Edge $\{d, e\}$ has the smallest weight of all these edges, so we add it to T. Now T has four edges, and we terminate. Note that we found the same T by using Kruskal's Algorithm.

Theorem 13.2 If G is a connected network on n vertices, Prim's Algorithm will terminate with a minimum spanning tree T of $n - 1$ edges. If G is a disconnected network, the algorithm will terminate with the message that G is disconnected because it is impossible to add another edge to T.

The proof is left as an exercise (Exercise 14).

As of this time, the best implementations of Prim's Algorithm run in $O(e + n \log_2 n)$ steps. (See Ahuja, Magnanti, and Orlin [1993], Graham and Hell [1985], Johnson [1975], or Kershenbaum and Van Slyke [1972]). See Gabow, *et al.* [1986] for a variant of Prim, that is currently the fastest algorithm for finding a minimum spanning tree. Another, early, unpublished minimum spanning tree algorithm is due to Sollin (see Ahuja, Magnanti, and Orlin [1993]). Yao [1975] improved on Sollin's Algorithm to develop an $O(e \log\log n)$ minimum spanning tree algorithm. See Exercise 16 for another minimum spanning tree algorithm that originates from work of Borůvka [1926a,b] and that applies if all weights are distinct.

EXERCISES FOR SECTION 13.1

1. For each network of Figure 13.2, find a minimum spanning tree using Kruskal's Algorithm.
2. Repeat Exercise 1 with Prim's Algorithm. Start with vertex a.
3. Apply Kruskal's Algorithm to each network of Figure 13.3.
4. Apply Prim's Algorithm to each network of Figure 13.3. Start with vertex a.

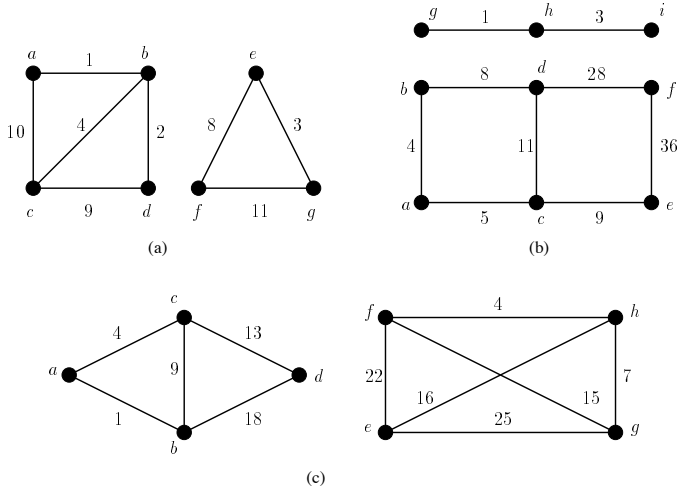


Figure 13.3: Networks for exercises of Section 13.1.

Table 13.1: Distance Between Vats

Vat	1	2	3	4	5	6	7	8
1	0	3.9	6.3	2.7	2.1	5.4	6.0	4.5
2	3.9	0	2.7	5.4	3.6	7.8	6.9	3.3
3	6.3	2.7	0	7.8	5.1	7.5	5.7	3.0
4	2.7	5.4	7.8	0	2.1	4.8	4.5	2.7
5	2.1	3.6	5.1	2.1	0	2.7	3.3	2.4
6	5.4	7.8	7.5	4.8	2.7	0	1.8	3.0
7	6.0	6.9	5.7	4.5	3.3	1.8	0	1.5
8	4.5	3.3	3.0	2.7	2.4	3.0	1.5	0

5. A chemical company has eight storage vats and wants to develop a system of pipelines that makes it possible to move chemicals from any vat to any other vat. The distance in feet between each pair of vats is given in Table 13.1. Determine between which pairs of vats to build pipelines so that chemicals can be moved from any vat to any other vat and so that the total length of pipe used is minimized.

6. A digital computer has a variety of components to be connected by high-frequency circuitry (wires) (see Example 3.30). The distance in millimeters between each pair of components is given in Table 13.2. Determine which pairs of components to connect so that the collection of components is connected and the total length of wire between components is minimized (to reduce capacitance and delay line effects).

7. For the set of points in Figure 13.4, use the method of Example 13.1 to find all partitions into 8 clusters, into 7 clusters, ..., into 1 cluster based on actual distance between the points.

Table 13.2: Distance Between Components

	1	2	3	4	5	6
1	0	6.7	5.2	2.8	5.6	3.6
2	6.7	0	5.7	7.3	5.1	3.2
3	5.2	5.7	0	3.4	8.5	4.0
4	2.8	7.3	3.4	0	8.0	4.4
5	5.6	5.1	8.5	8.0	0	4.6
6	3.6	3.2	4.0	4.4	4.6	0

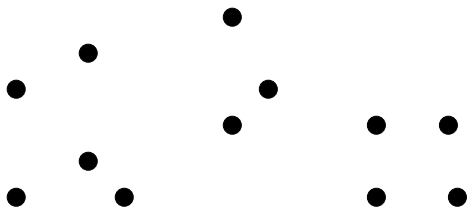


Figure 13.4: A set of points.

8. Using straight-line distances between cities, find the minimum spanning tree that connects Washington, DC and each (continental) U.S. state capital. (These distances can be found easily on the Internet.)
9. In each network of Figure 13.2, find a *maximum spanning tree*, a spanning tree so that no other spanning tree has a larger sum of weights.
10. Modify Kruskal's Algorithm so that it finds a maximum spanning tree.
11. If each edge of a network has a different weight, can there be more than one minimum spanning tree? Why?
12. (a) In the network of Figure 13.1, find a minimum spanning tree if it is required that the tree include edge $\{a, e\}$. (Such a problem might arise if we are required to include a particular telephone line or if one already exists.)

    (b) Repeat part (a) for edge $\{c, e\}$.
13. How would you modify Kruskal's Algorithm to deal with a situation such as that in Exercise 12, where one or more edges are specified as having to belong to the spanning tree?
14. Prove that Prim's Algorithm works.
15. Show by a crude argument that the computational complexity of Prim's Algorithm is bounded by a polynomial in e or in n.
16. Another algorithm for finding a minimum spanning tree T in an n vertex network G, provided that all weights are distinct, works as follows.

    Step 1. Set $T = \emptyset$.

    Step 2. Let G' be the spanning subgraph of G consisting of edges in T.

    Step 3. For each connected component K of G', find the minimum-weight edge of G joining a vertex of K to a vertex of some other component of G'. If there is

no such edge, stop and output the message that G is disconnected. Otherwise, add all the new edges to T.

Step 4. If T has $n-1$ edges, stop and output T. Otherwise, repeat Step 2.

This algorithm has its origin in the work of Borůvka [1926a,b].

(a) Apply this algorithm to each network of Figure 13.2.

(b) Prove that the algorithm works provided that all weights are distinct.

17. Show that a minimum spanning tree T is unique if and only if any edge $\{x, y\}$ not in T has larger weight than any edge on the circuit created by adding edge $\{x, y\}$ to T.

18. Let G be a connected graph. A *simple cut set* in G is a set B of edges whose removal disconnects G but such that no proper subset of B has this property. Let F be a set of edges of G. Show the following.

(a) If F is a simple cut set, F satisfies the following property C: For every spanning tree H of G, some edge of F is in H.

(b) If F satisfies property C but no proper subset of F does, then F is a simple cut set.

19. (Ahuja, Magnanti, and Orlin [1993]) How would you find a spanning tree T that minimizes

$$\left[\sum_{\{i,j\}\in T} (w_{ij})^2\right]^{1/2},$$

where w_{ij} is the weight on edge $\{i, j\}$?

20. (Ahuja, Magnanti, and Orlin [1993]) Two spanning trees T and T' are *adjacent* if they have all but one edge in common. Show that for any two spanning trees T' and T'', we can find a sequence of spanning trees

$$T' = T_1,\ T_2,\ T_3,\ \ldots,\ T_k = T'',$$

with T_i adjacent to T_{i+1} for $i = 1, 2, \ldots, k-1$.

21. (Ahuja, Magnanti, and Orlin [1993]) Suppose in network G that every edge is colored either red or blue.

(a) Show how to find a spanning tree with the maximum number of red edges.

(b) Suppose that some spanning tree has k' red edges and another has $k'' > k'$ red edges. Show that for k with $k'' > k > k'$, there is a spanning tree with k red edges.

13.2 THE SHORTEST ROUTE PROBLEM

13.2.1 The Problem

In this section we consider the problem of finding the shortest route between two vertices in a (directed) network. We begin with some examples that illustrate the problem.

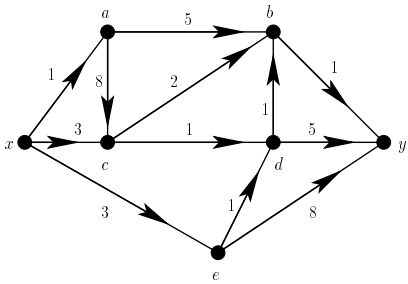


Figure 13.5: A shortest path from x to y is x, c, d, b, y.

Example 13.2 Interstate Highways Suppose that you wish to drive from New York to Los Angeles using only interstate highways. What is the shortest route to take? This problem can be translated into a network problem as follows. Let the vertices of a graph be junctions of interstate highways and join two such vertices by an edge if there is a single interstate highway joining them and uninterrupted by junctions. Put a real number on the edge $\{x, y\}$ representing the number of miles by interstate highway between vertices x and y. In the resulting network, let the *length* of a chain be defined to be the sum of the numbers (weights) on its edges, and take the *distance* $d(x, y)$ between vertices x and y to be the length of the shortest chain between x and y. (If there is no chain between x and y, distance is undefined.) We seek that chain between New York and Los Angeles that is of minimum length, that is, of length equal to d(New York, Los Angeles). ■

Example 13.3 Planning an Airplane Trip Suppose that you wish to fly from New York to Bangkok. What is the route that will have you spending as little time in the air as possible? To answer this question, let the vertices of a digraph be cities in the world air transportation network, and draw an arc from x to y if there is a direct flight from city x to city y. Put a real number on arc (x, y) representing the flying time. Then define the *length* of a path from x to y in this directed network as the sum of the numbers (weights) on the arcs, and define the *distance* $d(x, y)$ from x to y as the length of the shortest path from x to y. (Distance is again undefined if there is no path from x to y.) We seek the path of shortest length from New York to Bangkok, that is, the path with length equal to d(New York, Bangkok). ■

In general, we deal with a network or a directed network, and we seek the shortest chain or path from vertex x to vertex y. We concentrate on directed networks. In Section 13.2.2 we present an algorithm for finding the shortest path from x to y in a directed network. First, let us illustrate the basic ideas. In the directed network of Figure 13.5, the path x, a, b, y has length $1 + 5 + 1 = 7$. The path x, c, d, b, y is shorter; its length is $3 + 1 + 1 + 1 = 6$. Indeed, this is a shortest path from x to y. Hence, $d(x, y) = 6$. Note that we say that x, c, d, b, y is *a* shortest path from x to y. There can be more than one path from x to y of length equal to $d(x, y)$. Here, x, e, d, b, y is another such path. Note also that $d(y, x)$ is undefined; there is no path from y to x.

The problem of finding a shortest path from vertex x to vertex y in a (directed) network is a very common combinatorial problem. According to Goldman [1981], it is perhaps the most widely encountered combinatorial problem in government. Goldman estimated that the shortest path algorithm developed by just one government agency, the Urban Mass Transit Agency in the U.S. Department of Transportation, was regularly applied *billions* of times a year.

Surprisingly, a wide variety of problems can be restated as shortest route problems. We give a few examples.

Example 13.4 The TEX Document Processing System (Ahuja, Magnanti, and Orlin [1993]) A widely used document processing system in the mathematical sciences, TEX, decomposes paragraphs into lines that are both left- and right-justified and so that the appearance is "optimized." (Note that the last line of a paragraph need only be left-justified.) To see how this is done, let the paragraph have words 1, 2, ..., n, in that order. Let $c_{i,j}$ denote the "unattractiveness" of a line that begins with word i and ends with word $j-1$ ($c_{i,j} \geq 0$). For example, $c_{i,j}$ could measure the absolute difference between the paragraph's formatted width and the total length of the words $i, i+1, \ldots, j-1$. The problem is to decompose a paragraph into lines so that the total "cost" of the paragraph is minimized. To see how this is a shortest route problem, let the vertices of a directed network be the words plus a "dummy" vertex (word) denoted $n+1$. Include arcs from i to all $j > i$. The cost on arc (i, j) is $c_{i,j}$. If $1 = i_1, i_2, \ldots, i_k = n+1$ is a shortest path from word 1 to word $n+1$, this means that the most attractive appearance will be obtained by using lines starting at words i_p, $p = 1, 2, \ldots, k-1$, and ending at words $i_{p+1} - 1$, for $p = 1, 2, \ldots, k-1$. (Note that vertex $n+1$ must be included so that the definition of how each line ends can include word n.)[4] ■

Example 13.5 An Inventory of Structural Steel Beams (Ahuja, Magnanti, and Orlin [1993], Frank [1965]) A construction company needs structural steel beams of various lengths. To save storage space and the cost of maintaining a large inventory, the company will cut longer beams into shorter lengths rather than keep beams of all desired lengths. What lengths of beams to keep in inventory will depend on the demand for beams of different lengths. Moreover, the cutting operation will waste some steel. How can the company determine what length beams to keep in inventory so as to minimize the total cost of setting up the inventory and of discarding usable steel lost in cutting? We can formulate this as a shortest route problem by defining a directed network with vertices $0, 1, \ldots, n$ corresponding to the different beam lengths that might be needed. We assume that a beam of length L_i is shorter than a beam of length L_{i+1} and a beam of length L_0 has length 0. We include an arc from vertex i to each vertex $j > i$. We will interpret the arc (i, j) as corresponding to the strategy of keeping beams of lengths L_j in inventory and using them to meet the demand for beams of length L_{i+1}, L_{i+2}, ..., L_j. A path from vertex 0 to vertex n then corresponds to a set of beam lengths

[4] Allowing hyphenated words to come at the end of a line is a complication that we have disregarded. In Exercise 27 we ask the reader to address this complication.

to keep in inventory. Suppose that D_i represents the demand for steel beams of length L_i. The cost $C_{i,j}$ on the arc (i, j) is given by

$$C_{i,j} = K_j + C_j \sum_{k=i+1}^{j} D_k, \tag{13.1}$$

where K_j is the cost of setting up the inventory facility to handle beams of length L_j, C_j is the cost of a beam of length L_j, and the second term corresponds to the cost of using beams of length L_j to meet the demand for all beams of lengths L_{i+1} to L_j. A shortest path from vertex 0 to vertex n gives us the assortment of beams to keep in inventory if we want to minimize total cost. Note that it takes into account the (presumed) higher cost of longer beams and thus the waste from cutting them. What oversimplifications does this analysis make? [See Exercise 20.] ■

There are numerous other examples of shortest route problems. We give several others in Section 13.2.3 and many more are discussed or cited in Ahuja, Magnanti, and Orlin [1993].

13.2.2 Dijkstra's Algorithm

The shortest path algorithm we present is due to Dijkstra [1959]. We present it for a directed network D. Let $w(u, v)$ be the weight on arc (u, v). We recall our standing assumption that weights on arcs in networks are nonnegative. We will need this assumption. It will also be convenient to let $w(u, v)$ be ∞ if there is no arc from u to v. The basic idea is that we find at the kth iteration the k vertices $u_1, u_2, \ldots, u_k$ that are the k closest vertices to x in the network, that is, the ones that have (up to ties) the k smallest distances $d(x, u_1), d(x, u_2), \ldots, d(x, u_k)$. We also find for each u_j a shortest path from x to u_j. Having solved this problem for k, we solve it for $k + 1$.

The general idea in going from the kth iteration to the $(k+1)$st iteration is that for each vertex v not among the k closest vertices, we calculate $d(x, u_j) + w(u_j, v)$ for all j, and find the $(k+1)$st closest vertex as a v for which this sum is minimized. The rationale for this is that if $x, a_1, a_2, \ldots, a_p, v$ is a shortest path from x to v, then $x, a_1, a_2, \ldots, a_p$ must be a shortest path from x to a_p. If not, we could find a shorter path from x to v by using such a shorter path from x to a_p. Moreover, if weights are all positive, if v is the $(k + 1)$st closest vertex to x, and $x, a_1, a_2, \ldots, a_p, v$ is a shortest path from x to v, then a_p must be among the k closest vertices. Even if weights can be zero, the $(k + 1)$st vertex v can be chosen so that there is a shortest path $x, a_1, a_2, \ldots, a_p, v$ from x to v with a_p among the k closest vertices.

At each stage of Dijkstra's Algorithm, we keep a list of vertices included in the first k iterations—this defines a class W—and we keep a list of arcs used in the shortest paths from x to u_j—this defines a class B—and we keep a record of $d(x, u_j)$ for all j. We stop once y is added to W. We then use B to construct the path. We are now ready to present the algorithm more formally.

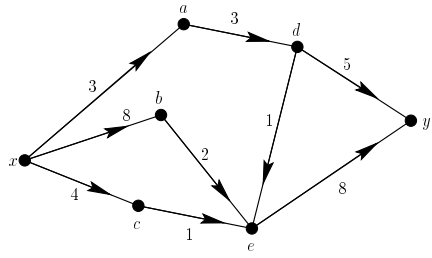


Figure 13.6: A directed network.

Algorithm 13.3: Dijkstra's Shortest Path Algorithm[5]

Input: A directed network D and vertices x and y from D.
Output: A shortest path from x to y or the message that there is no path from x to y.

Step 1. Initially, place vertex x in the class W, let $B = \emptyset$, and let $d(x, x) = 0$.

Step 2.

Step 2.1. For each vertex u in W and each vertex v not in W, let $\alpha(u, v) = d(x, u) + w(u, v)$. Find u in W and v not in W such that $\alpha(u, v)$ is minimized. (Choose arbitrarily in case of ties.)

Step 2.2. If the minimum in Step 2.1 is ∞, stop and give the message that there is no path from x to y.

Step 2.3. If the minimum in Step 2.1 is not ∞, place v in W (v is the next vertex chosen), place arc (u, v) in B, and set $d(x, v) = \alpha(u, v)$.

Step 3. If y is not yet in W, return to Step 2. If y is in W, stop. A shortest path from x to y can be found by using the unique path of arcs of B that goes from x to y. This can be found by working backward from y.

Let us illustrate this algorithm on the directed network of Figure 13.6. The successive steps in the algorithm are shown in Table 13.3. [The table does not show the $\alpha(u, v)$ that are infinite.] In iteration 2, for instance, arc (x, a) has the smallest α value of those computed, so a is added to W and (x, a) to B and $d(x, a)$ is taken to be $\alpha(x, a)$; in iteration 4, $\alpha(c, e)$ is minimum, so e is added to W and (c, e) to B, and $d(x, e)$ is taken to be $\alpha(c, e)$; and so on. At the seventh iteration, vertex y has been added to W. We now work backward from y, using arcs in B. We find that we got to y from d, to d from a, and to a from x. Thus, we find the path $x, (x, a), a, (a, d), d, (d, y), y$, that is a shortest path from x to y.

Again we comment on the computational complexity of the algorithm. The algorithm takes at most n iterations, where n is the number of vertices of the directed network. For each iteration adds a vertex. Each iteration involves additions $d(x, u) + w(u, v)$, one for each pair of vertices u in W and v not in W, that is, at

[5] At this point, we present the basic idea of the algorithm. Later, we describe how to improve it.

Table 13.3: Dijkstra's Algorithm Applied to the Directed Network of Figure 13.6

Iteration	Finite numbers $\alpha(u,v)$ computed	Added to W	Added to B	New $d(x,v)$
1	None	x	—	$d(x,x) = 0$
2	$\alpha(x,a) = 0+3 = 3$ $\alpha(x,b) = 0+8 = 8$ $\alpha(x,c) = 0+4 = 4$	a	(x,a)	$d(x,a) = 3$
3	$\alpha(x,b) = 0+8 = 8$ $\alpha(x,c) = 0+4 = 4$ $\alpha(a,d) = 3+3 = 6$	c	(x,c)	$d(x,c) = 4$
4	$\alpha(x,b) = 0+8 = 8$ $\alpha(a,d) = 3+3 = 6$ $\alpha(c,e) = 4+1 = 5$	e	(c,e)	$d(x,e) = 5$
5	$\alpha(x,b) = 0+8 = 8$ $\alpha(a,d) = 3+3 = 6$ $\alpha(e,y) = 5+8 = 13$	d	(a,d)	$d(x,d) = 6$
6	$\alpha(x,b) = 0+8 = 8$ $\alpha(e,y) = 5+8 = 13$ $\alpha(d,y) = 6+5 = 11$	b	(x,b)	$d(x,b) = 8$
7	$\alpha(e,y) = 5+8 = 13$ $\alpha(d,y) = 6+5 = 11$	y	(d,y)	$d(x,y) = 11$

most n^2 additions in all. Also, finding the smallest number among the at most n^2 numbers $d(x,u) + w(u,v)$ can be accomplished in at most n^2 comparisons. Hence, at each iteration the algorithm takes at most $2n^2$ steps. Altogether, the algorithm takes at most $2n^3$ steps, a polynomial bound. In the terminology of Section 2.18, we have an $O(n^3)$ algorithm.

Actually, by several simple changes in the procedure, the algorithm can be improved to be an $O(n^2)$ algorithm.[6] Suppose that we let u_k be the vertex that is the kth closest. We take $u_1 = x$. Let $\alpha_1(v) = \infty$ for all $v \neq x, \alpha_1(x) = 0$, and define

$$\alpha_{k+1}(v) = \min\{\alpha_k(v), \alpha_k(u_k) + w(u_k, v)\}. \tag{13.2}$$

Then it is easy to show (Exercise 25) that

$$\alpha_{k+1}(v) = \min\{d(x,u_j) + w(u_j,v) : j = 1,2,\ldots,k\}. \tag{13.3}$$

At each iteration, we compute the n numbers $\alpha_{k+1}(v)$. For each v we do this by first doing one addition and then finding one minimum of two numbers. Then we find the minimum of the set of numbers $\alpha_{k+1}(v)$, that can be done in at most n

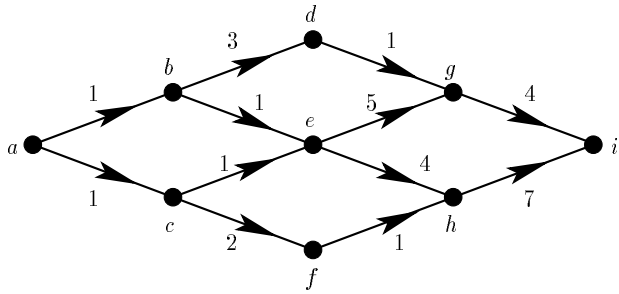
[6] The rest of this subsection may be omitted.

steps. The vertex v that gives us this minimum is u_{k+1}. (Choose arbitrarily in case of ties.) The total number of steps at the iteration that adds u_{k+1} is now at most $n + 2$, so the total number of steps in the entire algorithm is at most $n^2 + 2n$. If we just use the labels $\alpha_k(v)$, we will have to be more careful to compute the set B. Specifically, at each stage that $\alpha_{k+1}(v)$ decreases, we will have to keep track of the vertex u_j such that $\alpha_{k+1}(v)$ is redefined to be $d(x, u_j) + w(u_j, v)$. At the point when u_{k+1} is taken to be v, the corresponding (u_j, v) will be added to B. Details of the improved version of Dijkstra's Algorithm are left to the reader. For a discussion of various implementations of and improvements to Dijkstra's Algorithm under different assumptions, see Ahuja, Magnanti, and Orlin [1993].

13.2.3 Applications to Scheduling Problems

Although the shortest route problem has been formulated in the language of distances, the weights do not have to represent distances. Many applications of the shortest route algorithm apply to situations where the arcs correspond to activities of some kind and the weight on an arc corresponds to the cost of the activity. The problem involves finding a sequence of activities that begins at a starting point, accomplishes a desired objective, and minimizes the total cost. Alternatively, the weight is the time involved to carry out the activity and the problem seeks a sequence of activities that accomplishes the desired objective in a minimum amount of time. We encountered similar problems in Section 11.6.3. In these situations the network is sometimes called a PERT (Project Evaluation and Review Technique) network or a CPM (Critical Path Method) network. We illustrate these ideas with the following examples.

Example 13.6 A Manufacturing Process A manufacturing process starts with a piece of raw wood. The wood must be cut into shape, stripped, have holes punched, and be painted. The cutting must precede the hole punching and the stripping must precede the painting. Suppose that cutting takes 1 unit of time; stripping takes 1 unit of time; painting takes 2 units of time for uncut wood and 4 units of time for cut wood; and punching holes takes 3 units of time for unstripped wood, 5 units for stripped but unpainted wood, and 7 units for painted wood. The problem is to find the sequence of activities that will allow completion of the process in as short a period of time as possible. We can let the vertices of a directed network D represent stages in the manufacturing process, for example, raw wood; cut wood; cut and holes punched; cut, stripped, and holes punched; and so on. We take an arc from stage i to stage j if a single activity can take the process from stage i to stage j. Then we put a weight on arc (i, j) corresponding to the amount of time required to go from i to j. The directed network in question is shown in Figure 13.7. There are, for example, arcs from b to d and e, because cut wood can next either have holes punched or be stripped. The arc (b, d) has weight 3 because it corresponds to punching holes in unstripped wood. We seek a shortest path in this directed network from the raw wood vertex (a) to the stripped, cut, holes punched, and painted vertex (i). ■



Key:

a. raw wood	*d*. cut and holes punched	*g*. cut, holes punched, and stripped
b. cut	*e*. cut and stripped	*h*. cut, stripped, and painted
c. stripped	*f*. stripped and painted	*i*. stripped, cut, holes punched, and painted

Figure 13.7: Directed network for a manufacturing process. The vertices correspond to stages in the process, arcs to activities taking the process from stage to stage, and weights to times required for the activities.

Example 13.7 Inspection for Defective Products on a Production Line (Ahuja, Magnanti, and Orlin [1993], White [1969]) A production line has n stages and a batch of items is run through the production line at the same time. At each stage, defects might be introduced. Inspection at each stage is costly, but only inspecting at the end of stage n means that a lot of production time might be spent on continuing to produce items that already have defects. What is the optimal plan for inspecting for defective items? Let us assume that inspections are "perfect" in the sense of uncovering all defects and that defective items are unrepairable and are immediately discarded. Suppose that B is the size of the batches of items sent through the production line, a_i is the probability of producing a defect at stage i, p_i is the cost of manufacturing one item in stage i, $f_{i,j}$ is the fixed cost of inspecting the batch of items after stage j given that the last inspection was after stage i, and $g_{i,j}$ is the cost of inspecting one item after stage j given that the last inspection was after stage i. (Note that the costs of inspecting depend on when the last inspection was made since defects could have been introduced at all intervening stages.) We can calculate the expected number of nondefective items at the end of stage i, B_i, as

$$B_i = B \prod_{k=1}^{i} (1 - a_k).$$

To formulate our question as a shortest route problem, let the vertices of a directed network be the stages 1, 2, ..., n with an extra vertex 0 corresponding to the beginning, and let the arcs go from stage i to each stage $j > i$. A path corresponds to the plan of inspecting its vertices (except vertex 0). The cost we associate with arc (i, j) is given by

$$c_{i,j} = f_{i,j} + B_i g_{i,j} + B_i \sum_{k=i+1}^{j} p_k. \tag{13.4}$$

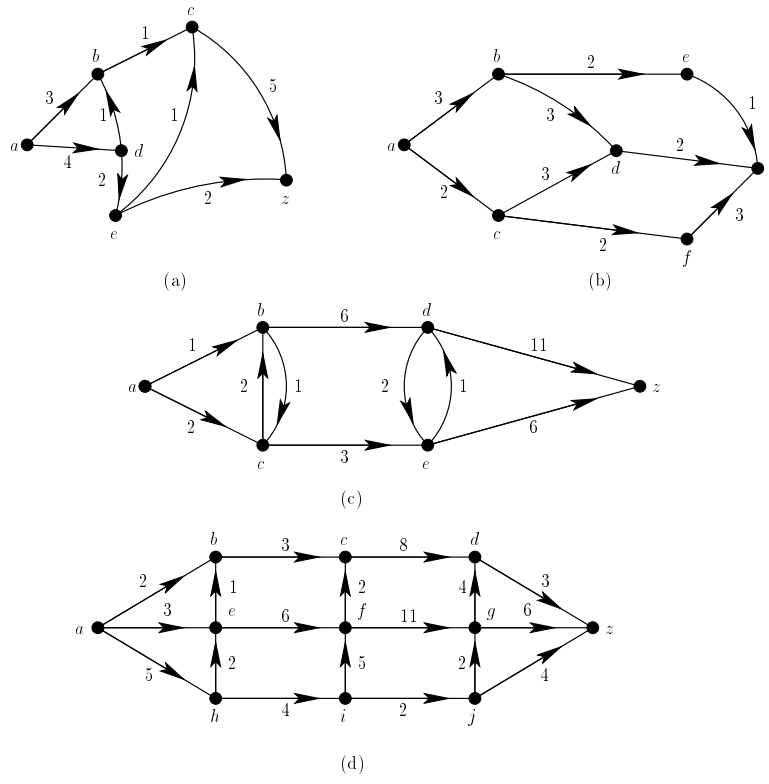


Figure 13.8: Directed networks for exercises of Sections 13.2 and 13.3.

This represents the total cost in stages $i+1, i+2, \ldots, j$ if we inspect in stage i and next in stage j. The first two terms are the fixed and variable costs of inspection after stage j, and the third term is the production cost in the stages $i+1$ through j. A shortest path from vertex 0 to vertex n gives us a least expensive inspection schedule. ■

EXERCISES FOR SECTION 13.2

1. Show that in a digraph, a shortest path from x to y must be a simple path.
2. Show that in a graph, a shortest chain from x to y must be a simple chain.
3. In each directed network of Figure 13.8, use Dijkstra's Algorithm (as described in Algorithm 13.3) to find a shortest path from a to z.
4. Find the most efficient manufacturing process in the problem of Example 13.6.
5. A product must be ground, polished, weighed, and inspected. The grinding must precede the polishing and the weighing, and the polishing must precede the inspection. Grinding takes 7 units of time, polishing takes 10 units of time, weighing takes

1 unit of time for an unpolished product and 3 units of time for a polished one, and inspection takes 2 units of time for an unweighed product and 3 units of time for a weighed one. What is the fastest production schedule?

6. A company wants to invest in a fleet of automobiles and is trying to decide on the best strategy for how long to keep a car. After 5 years it will sell all remaining cars and let an outside firm provide transportation. In planning over the next 5 years, the company estimates that a car bought at the beginning of year i and sold at the beginning of year j will have a net cost (purchase price minus trade-in allowance, plus running and maintenance costs) of a_{ij}. The numbers a_{ij} in thousands of dollars are given by the following matrix:

$$(a_{ij}) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 4 & 6 & 9 & 12 & 20 \\ & 5 & 7 & 11 & 16 \\ & & 6 & 8 & 13 \\ & & & 8 & 11 \\ & & & & 10 \end{pmatrix} \end{array}.$$

To determine the cheapest strategy for when to buy and sell cars, we let the vertices of a directed network be the numbers $1, 2, 3, 4, 5, 6$, include all arcs (i, j) for $i < j$, and let weight $w(i, j)$ be a_{ij}. The arc (i, j) has the interpretation of buying a car at the beginning of year i and selling it at the beginning of year j. Find the cheapest strategy.

7. In Example 13.5, suppose that 8 beam lengths are available. Each length corresponds to a vertex in a directed network N. The following chart contains the demand (D_j), inventory facility cost (K_j), and beam cost (C_j) for each of the beam lengths:

j	0	1	2	3	4	5	6	7
L_j	0	2	4	8	16	32	64	128
D_j	0	4	6	16	2	8	5	4
K_j	0	1	2	3	4	5	6	7
C_j	0	10	19	27	34	40	45	49

The costs $C_{i,j}$ for most of the arcs in N are given by the following matrix:

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} & 4 & 10 & 26 & X & 36 & 41 & 45 \\ & & 6 & 22 & 24 & 32 & 37 & 41 \\ & & & 16 & 18 & Y & 31 & 35 \\ & & & & 2 & 10 & 15 & Z \\ & & & & & 8 & 13 & 17 \\ & & & & & & 5 & 9 \\ & & & & & & & 4 \\ & & & & & & & \end{pmatrix} \end{array}.$$

(a) Using (13.1), compute the missing arc costs (X, Y, Z) in the matrix.

(b) Find a shortest path from vertex 0 to vertex 7 in the directed network N. Which beams should be kept in inventory?

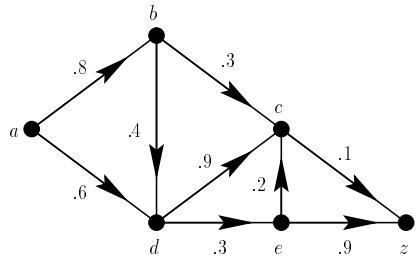


Figure 13.9: Communication network.

8. Figure 13.9 shows a communication network [arc (i, j) corresponds to a link over which i can communicate directly with j]. Suppose that the weight p_{ij} on the arc (i, j) is the probability that the link from i to j is operative. Assuming that defects in links occur independent of each other, the probability that all the links in a path are operative is the product of the link probabilities. Find the most reliable path from a to z. (*Hint:* Consider $-\log p_{ij}$.)

9. Suppose that you plan to retire in 20 years and want to invest \$100,000 toward retirement and you know that at year i, there will be available investment alternatives $A(i, 1)$, $A(i, 2)$, ..., $A(i, k_i)$, with investment $A(i, j)$ giving $r(i, j)$ rate of return and reaching maturity in $y(i, j)$ years. (This is, of course, very oversimplified.) Formulate the problem of finding an optimal investment strategy as a shortest route problem.

10. It is not efficient to consider all possible paths from i to j in searching for a shortest path. To illustrate the point, suppose that D has n vertices and an arc from every vertex to every other vertex. If x and y are any two vertices of D, find the number of paths from x to y.

11. If the cost of each arc increases by u units, does the cost of the shortest route from a to z increase by a multiple of u? Why?

12. In Example 13.4, suppose that we want to create a paragraph with 6 words. The following matrix shows the costs, $c_{i,j}$, for each i and j, $i < j \leq 7$. ("Word" 7 corresponds to the dummy vertex.)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} & 8 & 4 & 0 & 5 & 10 & 15 \\ & & 7 & 3 & 2 & 7 & 12 \\ & & & 7 & 3 & 3 & 8 \\ & & & & 10 & 6 & 9 \\ & & & & & 2 & 4 \\ & & & & & & 0 \\ & & & & & & \end{pmatrix} \end{array}$$

   (a) Build the network N whose vertices are the words (including "word" 7), whose arcs are (i, j), with $j > i$, and where each arc (i, j) has weight $c_{i,j}$.

   (b) Find the shortest path in N from word 1 to word 7.

   (c) Describe how the 6-word paragraph will be formatted.

13. In Example 13.7, suppose that there are 6 stages in a production line. The total cost of inspecting at stage i and next in stage j is given by Equation (13.4) using the following fixed, variable, and production cost equations:

$$\begin{aligned} f_{i,j} &= (j-i) \\ g_{i,j} &= (j-i)\frac{i}{j} \\ (p_k) &= (5,8,4,7,2,6) \\ B_i &= .8(1.1)^{-i} \end{aligned}$$

Find a least expensive production schedule.

14. (Bondy and Murty [1976]) A wolf, a goat, and a cabbage are on one bank of a river and a boatman will take them across, but can only take one at a time. The wolf and the goat cannot be left on one bank of the river together, nor can the goat and the cabbage. How can the boatman get them across the river in the shortest amount of time?

15. (Bondy and Murty [1976]) A man has a full 8-gallon jug of wine and empty jugs that hold 5 and 3 gallons, respectively. What is the fewest number of steps required for the man to divide the wine into two equal amounts?

16. (Ahuja, Magnanti, and Orlin [1993]) You own a summer rental property and at the beginning of the summer season, you receive a number of requests for the property. Each request gives you the days the rental would start and finish. Your rates for rentals of x days are given by a function $f(x)$. How do you decide which requests to accept in order to maximize your income for the summer? Formulate the problem as a shortest route problem.

17. (Ahuja, Magnanti, and Orlin [1993], Ravindran [1971]) A library wishes to minimize the cost of shelving for books in a special collection. A certain number of books in the collection have height H_i for $i = 1, 2, \ldots, n$, with $H_i < H_{i+1}$. We want to shelve books of the same height together and shelve books in increasing order of height. Assume that the books have known thickness. This then determines the total length L_i of shelving needed for books of height H_i. There is a fixed cost F_i of ordering shelves of height H_i that is independent of the length of shelving ordered, and there is a cost C_i per inch of length of shelving of height H_i. Thus, if we order shelving of height H_i of total length x_i, the cost of that shelving is $F_i + C_i x_i$. Note that since there is a fixed cost per order, we might not order shelving of each height since we can always shelve books of height H_i in shelving of height H_j for $j > i$. How much shelving of each length should we order? Formulate this as a shortest path problem.

18. Recall the problem of planning an airplane trip (Example 13.3). Besides the time spent in the air flying, consider the case where there is a "cost" associated with each city and it represents the average layover time at that city.

    (a) Formulate this "new" problem as a standard shortest path problem with only arc weights.

    (b) Solve this problem for the directed network of Figure 13.5 if the costs (average layover times) for each city are given by

City	a	b	c	d	e
Cost	1	2	5	4	3

.

19. If all arcs in a network have different weights, does the network necessarily have a unique shortest path? Give a proof or counterexample.

20. (Ahuja, Magnanti, and Orlin [1993]) Our formulation of the structural beam inventory problem in Example 13.5 has a variety of oversimplifications.

    (a) What does it assume happens with a beam of length 5 if it is cut to make a beam of length 2?

    (b) What other oversimplifications are there in this example?

    (c) If we can cut a single beam into multiple beams of the same length (e.g., a beam of length 14 into three beams of length 4, with some wastage), how might we modify the analysis?

    (d) How might we modify the analysis if we can not only cut a beam into multiple beams of the same length but can also sell the leftover for scrap value, measured say at $\$d$ per unit length?

21. Let D be a digraph. We define distance $\bar{d}(x, y)$ to be the length of the shortest path from x to y in the directed network obtained from D by putting a weight of 1 on each arc. Discuss how to use the powers of the adjacency matrix of D (Section 3.7) to calculate $\bar{d}(x, y)$.

22. A shortest path from a to z in a network is not necessarily unique, nor does it have the smallest number of arcs in a path from a to z. However, among all shortest paths from a to z, does Dijkstra's Algorithm produce one with the smallest number of arcs? Why?

23. In Dijkstra's Algorithm, show that if direction of arcs is disregarded, the arcs of the set B define a tree rooted at vertex x.

24. Describe a shortest path algorithm for finding, in a directed network D, the shortest paths from a given vertex x to each other vertex.

25. Show that α_{k+1} defined by (13.2) satisfies (13.3).

26. (a) Write a careful description of the $O(n^2)$ version of Dijkstra's Algorithm using the labels $\alpha_k(v)$.

    (b) Apply the algorithm to each directed network of Figure 13.8 to find a shortest path from a to z.

27. In Example 13.4 and Exercise 12, we did not consider the possibility of hyphenating a word to better optimize a paragraph's appearance. How could you modify the directed network formulation of the document processing system in Example 13.4 to accommodate hyphenation?

13.3 NETWORK FLOWS

13.3.1 The Maximum-Flow Problem

Suppose that D is a directed network and let $c_{ij} = w(i, j)$ be the nonnegative weight on the arc (i, j). In this section we call c_{ij} the *capacity* of the arc (i, j) and interpret it as the maximum amount of some commodity that can "flow" through

the arc per unit time in a steady-state situation. The commodity can be finished products, messages, people, oil, trucks, letters, electricity, and so on.

Flows are permitted only in the direction of the arc, that is, from i to j. We fix a *source* vertex s and a *sink* vertex t, and think of a flow starting at s and ending at t. (In all of our examples, s will have no incoming arcs and t no outgoing arcs. But it is not necessary to assume these properties.) Let x_{ij} be the flow through arc (i, j). Then we require that

$$0 \leq x_{ij} \leq c_{ij}. \tag{13.5}$$

This says that flow is nonnegative and cannot exceed capacity. We also have a *conservation law*, which says that for all vertices $i \neq s, t$, what goes in must go out; that is,

$$\sum_j x_{ij} = \sum_j x_{ji}, \quad i \neq s, t. \tag{13.6}$$

A set of numbers $x = \{x_{ij}\}$ satisfying (13.5) and (13.6) is called an (s, t)-*feasible flow*, or an (s, t)-*flow*, or just a *flow*. For instance, consider the directed network of Figure 13.10. In part (a) of the figure, the capacities c_{ij} are shown on each arc with the numbers in squares. In part (b), a flow is shown with the numbers in circles. Note that (13.5) and (13.6) hold. For instance, $c_{24} = 2$ and $x_{24} = 0$, so $0 \leq x_{24} \leq c_{24}$. Also, $\sum_j x_{3j} = x_{35} + x_{36} + x_{37} = 2 + 1 + 0 = 3$, and $\sum_j x_{j3} = x_{23} = 3$; and so on.

Suppose that x defines a flow. Let

$$v_t = \sum_j x_{jt} - \sum_j x_{tj}$$

and

$$v_s = \sum_j x_{js} - \sum_j x_{sj}.$$

Note that if we sum the terms

$$\sum_j x_{ji} - \sum_j x_{ij}$$

over all i, Equation (13.6) tells us that all of these terms are 0 except for the cases $i = s$ and $i = t$. Thus,

$$\sum_i \left[\sum_j x_{ji} - \sum_j x_{ij} \right] = v_t + v_s. \tag{13.7}$$

However, the left-hand side of (13.7) is the same as

$$\alpha = \sum_{i,j} x_{ji} - \sum_{i,j} x_{ij}. \tag{13.8}$$

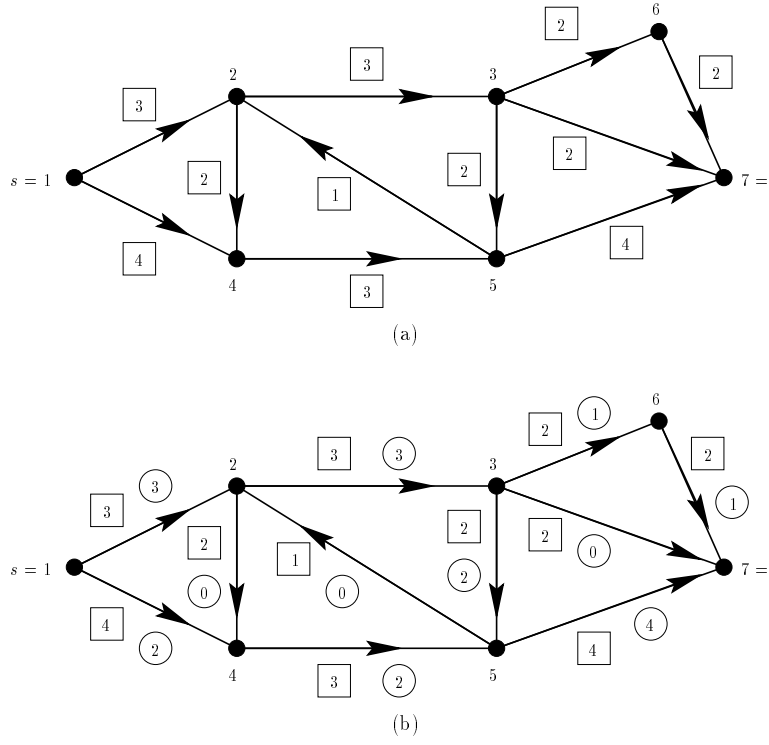


Figure 13.10: Capacities are shown in squares, flows in circles.

Since both sums in (13.8) simply sum all flows over all arcs, they are equal and hence $\alpha = 0$. We conclude that $v_s = -v_t$. Thus, there is a number v such that

$$\sum_j x_{ji} - \sum_j x_{ij} = \begin{cases} -v & \text{if } i = s \\ v & \text{if } i = t \\ 0 & \text{if } i \neq s, t. \end{cases} \tag{13.9}$$

The number v in our example of Figure 13.10(b) is 5. It is called the *value* of the flow. The value represents the total amount of the commodity that can be sent through the network in a given period of time if this flow is used. The problem we will consider is this: Find a flow that has maximum value, a *maximum flow*.

The classic reference on the theory of flows in networks is the book by Ford and Fulkerson [1962]. Other comprehensive references are Ahuja, Magnanti, and Orlin [1993], Berge and Ghouila-Houri [1965], Cook, *et al.* [1998], Frank and Frisch [1971], Hu [1969], Iri [1969], Lawler [1976], Minieka [1978], and Papadimitriou and Steiglitz [1982].

Although our presentation is for directed networks, everything in this section will apply to undirected networks. Simply replace each (undirected) edge $\{i, j\}$ with capacity c_{ij} by two arcs (i, j) and (j, i) and let each of these arcs have the same capacity as edge $\{i, j\}$.

A large number of combinatorial optimization problems can be formulated as network flow problems. The next example illustrates this point. For many practical applications of network flows, the reader should consult such books and survey articles as Ahuja, Magnanti, and Orlin [1993], Aronson [1989], Bazaraa, Jarvis, and Sherali [1990], Glover and Klingman [1977], and Gondran and Minoux [1984].

Example 13.8 College Admissions The admissions officer at a university has a list of applicants. Each applicant has provided a list of potential majors that she or he might consider. The job of the admissions officer is to admit a first round of applicants so that every major represented among the applicants' interests has exactly one student admitted and so that the number of students admitted from state i is no more than q_i. Can this be done? Build a directed network whose vertices are the applicants, the majors, and the states, plus a source vertex s and a sink vertex t. Include arcs (s, m) for every major m, (m, a) whenever applicant a is interested in major m, (a, i) if applicant a lives in state i, and (i, t) for every state i. Let arc (i, t) have capacity q_i and all other arcs have capacity 1. We seek a maximum flow from s to t in this network. If the maximum flow value equals the number of majors represented among the applicants' interests, it is possible to solve the problem. If not, it cannot. The proof is left to the reader (Exercise 31). ■

13.3.2 Cuts

Let S and T be two sets that partition the vertex set of the digraph D, that is, $V(D) = S \cup T$ and $S \cap T = \emptyset$. We refer to the partition (S, T) as a *cut*. Equivalently, we think of the cut as the set C of all arcs that go from vertices in S to vertices in T. The set C is called a cut because after the arcs of C are removed, there is no path from any vertex of S to any vertex of T. If x is any vertex of S and y any vertex of T, C is called an (x, y)-*cut*. For instance, in Figure 13.10, $S = \{1, 2\}$ and $T = \{3, 4, 5, 6, 7\}$ is a cut, and this is equivalent to the set of arcs $\{(1, 4), (2, 3), (2, 4)\}$. [We do not include arc $(5, 2)$ as it goes from a vertex in T to a vertex in S.] Notice that if there are more than two vertices, there are always at least two (x, y)-cuts: $S = \{x\}, T = V(D) - \{x\}$, and $S = V(D) - \{y\}, T = \{y\}$.

In a directed network, if (S, T) is a cut, we can define its *capacity* as $c(S, T) = \sum_{i \in S} \sum_{j \in T} c_{ij}$. In our example above, $c(S, T) = c_{14} + c_{23} + c_{24} = 9$. Notice that in our example, the value of the flow is 5 and the capacity of this cut is 9, which is greater. The next result shows that this is no accident.

Theorem 13.3 In a directed network, the value of any (s, t)-flow is $\leq$ the capacity of any (s, t)-cut.

Proof. Let x be an (s, t)-flow and (S, T) be an (s, t)-cut. Note that $\sum_j x_{ij} - \sum_j x_{ji}$ is 0 if $i \in S$ and $i \neq s$, and is v if $i = s$. Thus,

$$v = \sum_j x_{sj} - \sum_j x_{js},$$

$$
\begin{aligned}
&= \sum_{i \in S} \left[\sum_j x_{ij} - \sum_j x_{ji} \right], \\
&= \sum_{i \in S} \sum_{j \in S} [x_{ij} - x_{ji}] + \sum_{i \in S} \sum_{j \in T} [x_{ij} - x_{ji}], \\
&= \sum_{i \in S} \sum_{j \in S} x_{ij} - \sum_{i \in S} \sum_{j \in S} x_{ji} + \sum_{i \in S} \sum_{j \in T} [x_{ij} - x_{ji}], \qquad (13.10) \\
&= \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}), \qquad (13.11)
\end{aligned}
$$

because the first two terms of (13.10) are the same. Thus, by (13.11), the value of any flow is the net flow through any cut. Since $x_{ij} \leq c_{ij}$ and $x_{ij} \geq 0$, we have

$$x_{ij} - x_{ji} \leq x_{ij} \leq c_{ij},$$

so (13.11) implies that

$$v \leq \sum_{i \in S} \sum_{j \in T} c_{ij} = c(S, T).$$

Q.E.D.

Corollary 13.3.1 In a directed network, if (S, T) is an (s, t)-cut and x is an (s, t)-flow, then

$$v = \sum_{i \in S} \sum_{j \in T} [x_{ij} - x_{ji}].$$

Proof. This is a corollary of the proof. Q.E.D.

Figure 13.11 shows another (s, t)-flow in the network of Figure 13.10(a). This flow has value 6. Notice that if $S = \{1, 2, 4\}$ and $T = \{3, 5, 6, 7\}$, then $c(S, T) = c_{23} + c_{45} = 6$. Now there can be no (s, t)-flow with value more than the capacity of this cut, that is, 6. Hence, the flow shown is a maximum flow. Similarly, this cut must be an (s, t)-cut of minimum capacity, a *minimum cut*. Indeed, the same thing must be true any time we find a flow and a cut where the value of the flow is the same as the capacity of the cut.

Theorem 13.4 If x is an (s, t)-flow with value v and (S, T) is an (s, t)-cut with capacity $c(S, T)$, and if $v = c(S, T)$, then x is a maximum flow and (S, T) is a minimum cut.

We have used reasoning similar to this in Theorem 12.4, where we argued that since the number of edges in any matching of a graph is less than or equal to the number of vertices in any covering, if we ever find a matching and a covering of the same size, the matching must be maximum and the covering must be minimum.

As with flows, we have emphasized cuts in directed networks in this section. However, we can speak of cuts in undirected networks N. As for directed networks, a *cut* in N is a partition of the vertices into two sets S and T or, equivalently, the

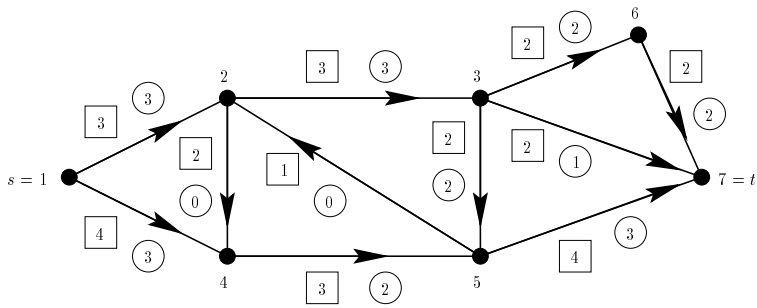


Figure 13.11: Another flow for the directed network of Figure 13.10(a).

set of all edges going from a vertex in S to a vertex in T. If $s \in S$ and $t \in T$, then (S, T) is called an (s, t)-*cut*.

Finding the minimum cut in a network is often of interest in its own right, as the next example shows.

Example 13.9 Distributed Computing on a Two-Processor Computer (Ahuja, Magnanti, and Orlin [1993], Stone [1977]) In the simplest version of a distributed computing system, there are two processors in the system and we wish to assign different modules (subroutines) of a program to the processors in such a way that minimizes the costs of interprocessor communication and computation. The cost of executing module i on processor 1 is a_i and on processor 2 is b_i. There is a relatively high interprocessor communication cost $c_{i,j}$ if modules i and j are assigned to different processors and these modules interact. We seek to allocate modules of the program to the two processors so as to minimize the total cost, which is the sum of the processing cost and the interprocessor communication cost. Build an undirected network N with a source s representing processor 1 and sink t representing processor 2 and a vertex for every module of the program. Add edges $\{s, i\}$ for every module i with capacity b_i, and edges $\{i, t\}$ for every module i with capacity a_i. Finally, if modules i and j interact during program execution, add an edge $\{i, j\}$ with capacity $c_{i,j}$. Let $E(N)$ be the set of edges in N. We note that there is a one-to-one correspondence between assignments of modules to processors and (s, t)-cuts and that the capacity of a cut is the same as the cost of the corresponding assignment. To see this, suppose that A_1 consists of all the modules assigned to processor 1 and similarly A_2. The cost of this assignment is

$$\sum_{i \in A_1} a_i + \sum_{i \in A_2} b_i + \sum_{\substack{\{i,j\} \in E(N), \\ i \in A_1,\, j \in A_2}} c_{i,j}.$$

The (s, t)-cut corresponding to this assignment consists of the sets $\{s\} \cup A_1$, $\{t\} \cup A_2$. The cut thus contains edges $\{i, t\}$ with capacity a_i for $i \in A_1$, $\{s, i\}$ with capacity b_i for $i \in A_2$, and $\{i, j\}$ with capacity $c_{i,j}$ for i, j interacting modules with i in A_1 and j in A_2. This is exactly the cost of the assignment. Hence, the minimum-cost

assignment we are seeking corresponds to the minimum (s,t)-cut in the network N. (We note that this example oversimplifies the practical problem. In fact, the processing cost and the interprocessor communication cost are measured in different units and a major challenge in practice is to "scale" these costs so that they can be compared.[7]) ■

13.3.3 A Faulty Max-Flow Algorithm

Our goal is to describe an algorithm for finding the maximum flow. First we present an intuitive, although faulty, technique. If P is a simple path from s to t, we call it an (s,t)-*path* and we let the corresponding *unit flow* x^P be given by

$$x_{ij}^P = \begin{cases} 1 & \text{if arc } (i,j) \text{ is in } P \\ 0 & \text{otherwise.} \end{cases}$$

The idea is to add unit flows successively.

Let us say that an arc (i,j) is *unsaturated* by a flow x if $x_{ij} < c_{ij}$, and let us define the *slack* by $s_{ij} = c_{ij} - x_{ij}$. The basic point is that if θ is the minimum slack among arcs of P, we can add the unit flow x^P a total of θ times and obtain a new flow with value increased by θ. Here is the algorithm.

Algorithm 13.4: Max-Flow Algorithm: First Attempt

Input: A directed network with a source s and a sink t.
Output: A supposedly maximum (s,t)-flow x.

Step 1. Set $x_{ij} = 0$, all i,j.

Step 2.

> ***Step 2.1.*** Find an (s,t)-path P with all arcs unsaturated. If none exists, go to Step 3.
>
> ***Step 2.2.*** Compute the slack of each arc of P.
>
> ***Step 2.3.*** Compute θ, the minimum slack of arcs of P.
>
> ***Step 2.4.*** Redefine x by adding θ to the flow on arc (i,j) if (i,j) is in P. Return to Step 2.1.

Step 3. Stop with the flow x.

Let us apply this algorithm to the directed network of Figure 13.10(a). Figure 13.12 shows the successive flows defined by the following iterations. In the first iteration, we take $x_{ij} = 0$, all i,j. We then note that $P = 1,4,5,7$ is an (s,t)-path with all arcs unsaturated, and the corresponding θ is 3, for $s_{14} = c_{14} - x_{14} = 4 - 0 = 4, s_{45} = c_{45} - x_{45} = 3 - 0 = 3$, and $s_{57} = c_{57} - x_{57} = 4 - 0 = 4$. Hence, we increase x_{14}, x_{45}, and x_{57} by $\theta = 3$, obtaining the second flow in Figure 13.12.

[7] We thank David Roberts for this observation.

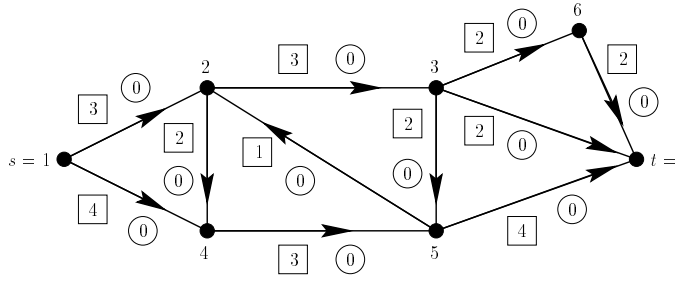
In this flow, $P = 1, 2, 3, 7$ is an (s, t)-path with all unsaturated arcs. Its θ is 2, for $s_{12} = 3, s_{23} = 3$, and $s_{37} = 2$. Thus, we increase x_{12}, x_{23}, and x_{37} by $\theta = 2$, obtaining the third flow in Figure 13.12. In this flow, $P = 1, 2, 3, 6, 7$ has all arcs unsaturated and $\theta = 1$, since $s_{12} = c_{12} - x_{12} = 3 - 2 = 1, s_{23} = 1, s_{36} = 2$, and $s_{67} = 2$. Thus, we increase x_{12}, x_{23}, x_{36}, and x_{67} by $\theta = 1$, obtaining the fourth flow in Figure 13.12. Since in this flow there are no more (s, t)-paths with all arcs unsaturated, we stop. Note that we have obtained a flow of value equal to 6, which we know to be a maximum.

Unfortunately, this algorithm does not necessarily lead to a maximum flow. Let us consider the same directed network. Figure 13.13 shows the successive steps in another use of this algorithm using different (s, t)-paths. Notice that after obtaining the fourth flow, we can find no (s, t)-path with all arcs unsaturated. Thus, the algorithm stops. However, the flow obtained has value only 5, which is not maximum. What went wrong?

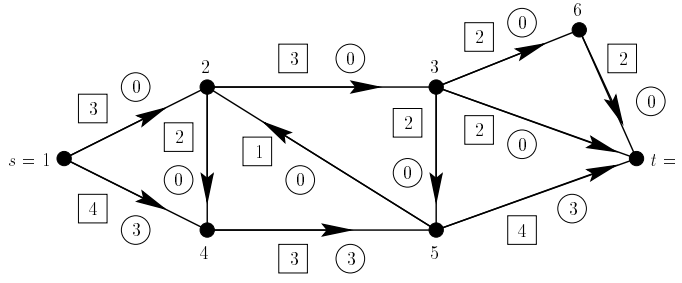
Consider the minimum cut $S = \{1, 2, 4\}, T = \{3, 5, 6, 7\}$. One problem is that one of the unit flow paths used, $1, 4, 5, 2, 3, 7$, crosses this cut backward, that is, from T to S; equivalently, it crosses it forward from S to T twice. Thus, the 1 unit of flow on P uses up 2 units of capacity—too much capacity is used. We would be better off if we got rid of some of the flow going backward and used more in a forward direction. This suggests a way to improve on our algorithm.

13.3.4 Augmenting Chains

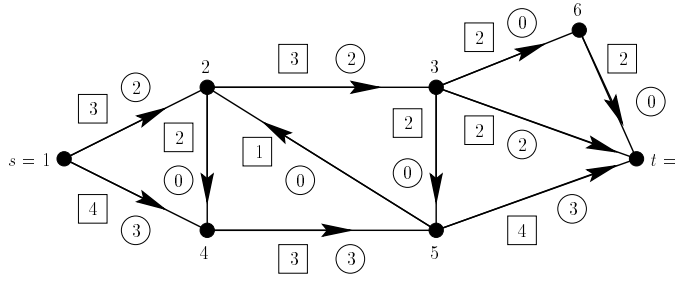
Consider the graph (multigraph) G obtained from a directed network by disregarding directions on arcs. Let C be a chain from s to t in this graph. An arc (i, j) of D is said to be a *forward arc* of C if it is followed from i to j, and a *backward arc* otherwise. For instance, in the directed network D of Figure 13.10, one chain is $1, 4, 5, 3, 6, 7$. Here, arc $(1, 4)$ is a forward arc, but arc $(3, 5)$ is backward. If x is a flow, C is said to be a *flow-augmenting chain*, or just an *augmenting chain*, *relative to* x if $x_{ij} < c_{ij}$ for each forward arc and $x_{ij} > 0$ for each backward arc. The unit flow paths with no saturated arcs, which we discussed in Section 13.3.3, all correspond to flow-augmenting chains with no backward arcs. The chain $1, 4, 5, 3, 6, 7$ in Figure 13.10(b) is a flow-augmenting chain relative to the flow shown in the figure. For the forward arcs $(1, 4), (4, 5), (3, 6)$, and $(6, 7)$ are all under capacity and the backward arc $(3, 5)$ has positive flow. We can improve on the value of the flow by decreasing the backward flow and increasing the forward flow. Specifically, we record x_{ij} for all backward arcs on the chain and compute and record the slack $s_{ij} = c_{ij} - x_{ij}$ for all forward arcs. If λ is the minimum of these recorded numbers, λ is called the *capacity* of the augmenting chain. We then increase each x_{ij} on a forward arc by λ and decrease each x_{ij} on a backward arc by λ. By choice of λ, each new x_{ij} is still nonnegative and is no higher than the capacity c_{ij}. Moreover, the conservation law (13.6) still holds. Next, we observe that the value of the flow increases by λ, for the chain starts with an edge from s and ends with an edge to t. If the edge from s is forward, the flow out of s is increased; if it is backward, the flow into s is decreased. In any case, the value or net flow out of s is increased. A



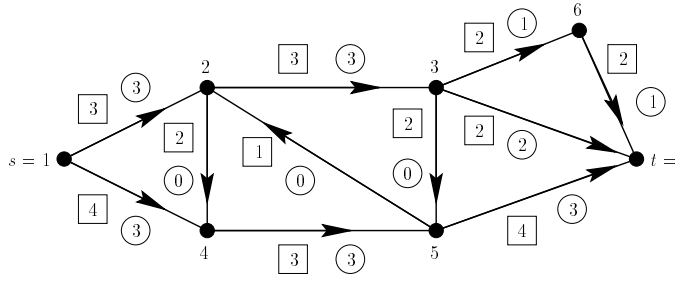
I



II (from $P = 1, 4, 5, 7$ and $\theta = 3$)

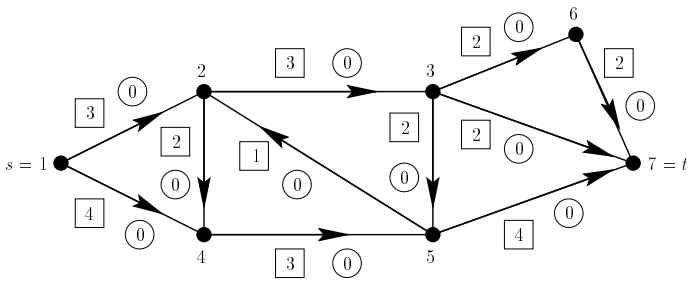


III (from $P = 1, 2, 3, 7$ and $\theta = 2$)

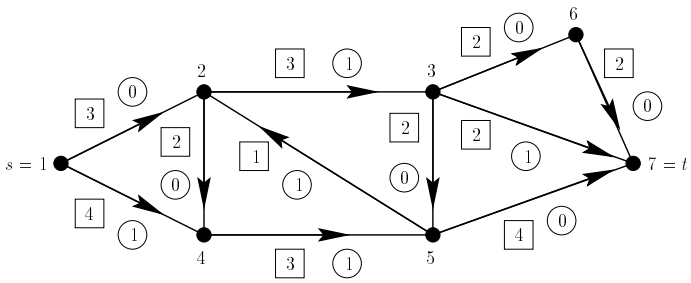


IV (from $P = 1, 2, 3, 6, 7$ and $\theta = 1$)

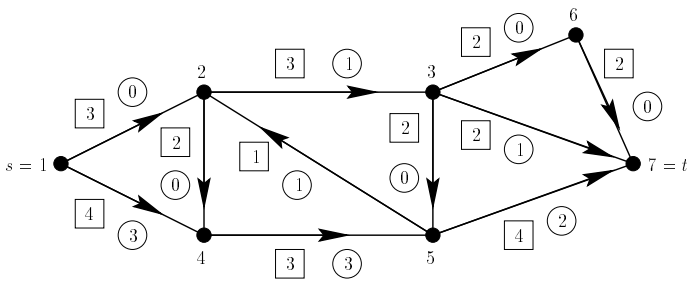
Figure 13.12: Applying Algorithm 13.4 to the directed network of Figure 13.10(a).



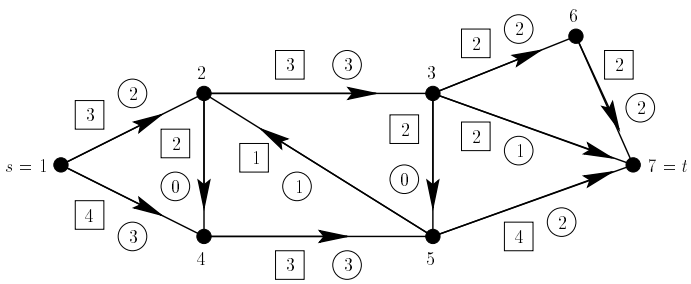
I



II (from $P = 1, 4, 5, 2, 3, 7$ and $\theta = 1$)



III (from $P = 1, 4, 5, 7$ and $\theta = 2$)



IV = (from P = 1, 2, 3, 6, 7 and T = 2)

Figure 13.13: Applying Algorithm 13.4 to the directed network of Figure 13.10(a) using a different choice of (s, t)-paths.

similar argument holds for the edge to t. [These arguments need to be expanded in case the flow-augmenting chain is not a simple chain. The expansion is left to the reader (Exercise 35).] However, we will only need simple flow-augmenting chains (Exercise 36). To illustrate with our example of Figure 13.10(b), note that $s_{14} = 2, s_{45} = 1, s_{36} = 1, s_{67} = 1$, and $x_{35} = 2$. Hence, the minimum of these numbers, λ, is 1. We increase x_{14}, x_{45}, x_{36}, and x_{67} by 1, and decrease x_{35} by 1, obtaining a flow of value 6, one more than the value of the flow shown.

We are now ready to state the main results about maximum flows, which will allow us to present our main algorithm. We have already shown that if a flow admits an augmenting chain, the value of the flow can be increased, so the flow is not maximum. The first result says that if the flow is not maximum, we can find an augmenting chain. (Thus, flow-augmenting chains are analogous to M-augmenting chains for matchings, and the next theorem is analogous to Theorem 12.7. We expand on the relation between network flows and matching in Section 13.3.8.)

Theorem 13.5 An (s,t)-flow is maximum if and only if it admits no augmenting chain from s to t.

Proof.[8] It remains to suppose that x is a flow with no augmenting chain and to show that x is a maximum. Let S be the set of vertices j such that there is an augmenting chain from s to j and let T be all other vertices. Note that s is in S because s alone defines an augmenting chain from s to s. Moreover, t is in T, since there is no augmenting chain. By definition of augmenting chain and by definition of S and T, we have for all i in S and all j in T, $x_{ij} = c_{ij}$ and $x_{ji} = 0$. For there is an augmenting chain from s to i, since i is in S. If $x_{ij} < c_{ij}$ or $x_{ji} > 0$, we can add edge $\{i,j\}$ to this chain to find an augmenting chain from s to j, contradicting j in T. Thus, for all i in S and j in T, $x_{ij} - x_{ji} = c_{ij}$.

By Corollary 13.3.1,

$$v = \sum_{i \in S} \sum_{j \in T} [x_{ij} - x_{ji}] = \sum_{i \in S} \sum_{j \in T} c_{ij} = c(S,T).$$

Hence, we have found a cut (S,T) with the same capacity as the value of the flow x. By Theorem 13.4, the flow is a maximum. Q.E.D.

A cut (S,T) is called *saturated* relative to a flow x if $x_{ij} = c_{ij}$ for all $i \in S, j \in T$, and $x_{ji} = 0$ for all $i \in S, j \in T$. The following is a corollary of the proof of Theorem 13.5.

Corollary 13.5.1 If (S,T) is a saturated (s,t)-cut relative to flow x, then x is a maximum (s,t)-flow.

The next result is a very famous theorem due to Elias, Feinstein, and Shannon [1956] and Ford and Fulkerson [1956].

[8] The proof may be omitted.

Theorem 13.6 (The Max-Flow Min-Cut Theorem) In a directed network, the maximum value of an (s,t)-flow equals the minimum capacity of an (s,t)-cut.

Proof.[9] It follows from Theorem 13.3 that the maximum value of an (s,t)-flow is at most the minimum capacity of an (s,t)-cut. To show equality, we suppose that x is a maximum flow, with value v. Then it can have no flow-augmenting chain, and so, by the proof of Theorem 13.5, we can find an (s,t)-cut (S,T) so that $v = c(S,T)$. It follows by Theorem 13.4 that x is a maximum flow and (S,T) is a minimum cut. Q.E.D.

Remark. Our proof of the Max-Flow Min-Cut Theorem uses the tacit assumption that there exists a maximum flow. This is easy to prove if all capacities are rational numbers. For then the Maximum-Flow Algorithm, described in Section 13.3.5, finds a maximum flow. If some capacities are not rational, a maximum flow still exists (even though the Maximum-Flow Algorithm, as we describe it, does not necessarily find a maximum flow). See Lawler [1976] or Papadimitriou and Steiglitz [1982] for a proof.

13.3.5 The Max-Flow Algorithm

We can now formulate the Max-Flow Algorithm.

Algorithm 13.5: The Max-Flow Algorithm

Input: A directed network with a source s and a sink t.
Output: A maximum (s,t)-flow x.

Step 1. Set $x_{ij} = 0$ for all i,j.

Step 2.

- ***Step 2.1.*** Find a flow-augmenting chain C from s to t. If none exists, go to Step 3.
- ***Step 2.2.*** Compute and record the slack of each forward arc of C and record the flow of each backward arc of C.
- ***Step 2.3.*** Compute λ, the minimum of the numbers recorded in Step 2.2.
- ***Step 2.4.*** Redefine x by adding λ to the flow on all forward arcs of C and subtracting λ from the flow on all backward arcs of C. Return to Step 2.1.

Step 3. Stop with the flow x.

Suppose that we apply this algorithm to the directed network of Figure 13.10(a). Since every (s,t)-path is an augmenting chain with no backward arcs, we can get to the fourth flow of Figure 13.13. We then identify the flow-augmenting chain

[9] The proof may be omitted.

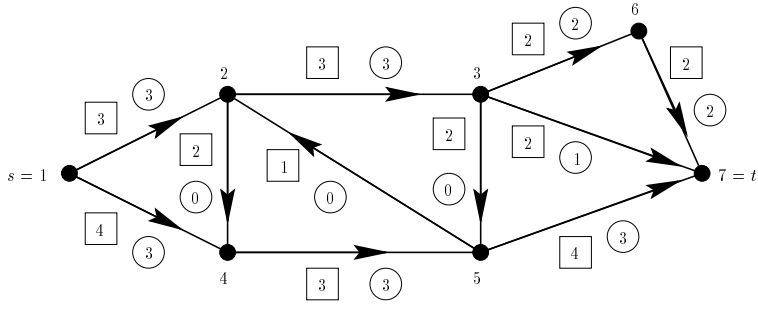


Figure 13.14: The flow obtained from the fourth flow of Figure 13.13 by using the flow-augmenting chain $C = 1, 2, 5, 7$.

$C = 1, 2, 5, 7$. In Step 2.2 we compute $s_{12} = 1, s_{57} = 2, x_{52} = 1$. Then the minimum, λ, of these numbers is $\lambda = 1$. We increase x_{12} and x_{57} by $\lambda = 1$ and decrease x_{52} by $\lambda = 1$, obtaining the flow of Figure 13.14. In this flow, there is no augmenting chain. We conclude that it is maximum. This agrees with our earlier conclusion, since the value of this flow is 6.

Theorem 13.7 If all capacities in the directed network are rational numbers, the Max-Flow Algorithm will attain a maximum flow.

Proof.[10] Clearly, if all capacities are integers, then at each iteration the number λ is an integer, and the value of the flow increases by λ. Since the value is at most the capacity of the cut defined by $S = \{s\}, T = V(D) - \{s\}$, the value cannot keep increasing by an integer amount more than a finite number of times. Hence, there will come a time when there is no augmenting chain from s to t. By Theorem 13.5, the corresponding flow will be maximum. The case of rational capacities can be handled by finding a common denominator δ for all the capacities and considering the directed network obtained from the original one by multiplying all capacities by δ. Q.E.D.

In general, if some capacities are not rational numbers, the algorithm does not necessarily converge in a finite number of steps.[11] Moreover, it can converge to a flow that is not a maximum, as Ford and Fulkerson [1962] showed. Edmonds and Karp [1972] have shown that if each flow augmentation is made along an augmenting chain with a minimum number of edges, the algorithm terminates in a finite number of steps and a maximum flow is attained. See Ahuja, Magnanti, and Orlin [1993], Lawler [1976], or Papadimitriou and Steiglitz [1982] for details. Ahuja, Magnanti, and Orlin [1989, 1991] give surveys of various improvements in network flow algorithms.

[10] The proof may be omitted.

[11] This point is of little practical significance, since computers work with rational numbers.

13.3.6 A Labeling Procedure for Finding Augmenting Chains[12]

The Max-Flow Algorithm as we have described it does not discuss how to find an augmenting chain in Step 2.1. In this subsection we describe a labeling procedure for finding such a chain. (The procedure is due to Ford and Fulkerson [1957].) In this procedure, at each step, a vertex is *scanned* and its neighbors are given *labels*. Vertex j gets label (i^+) or (i^-). The label is determined by finding an augmenting chain from s to j which ends with the edge $\{i, j\}$. The $+$ indicates that (i, j) is a forward arc in this augmenting chain, the $-$ that it is a backward arc. Eventually, if vertex t is labeled, we have an augmenting chain from s to t. If the procedure concludes without labeling vertex t, we will see that no augmenting chain exists, and we conclude that the flow is maximum. The procedure is described in detail as follows.

Algorithm 13.6: Subroutine: Labeling Algorithm for Finding an Augmenting Chain

Input: A directed network with a source s, a sink t, and an (s, t)-flow x.
Output: An augmenting chain or the message that x is a maximum flow.

Step 1. Give vertex s the label $(-)$.

Step 2. Let F be the set of arcs (i, j) such that $s_{ij} > 0$. Let B be the set of arcs (i, j) such that $x_{ij} > 0$. (Note that arcs in F can be used as forward arcs and arcs in B as backward arcs in an augmenting chain.)

Step 3. (*Labeling and Scanning*)

Step 3.1. If all labeled vertices have been scanned, go to Step 5.

Step 3.2. If not, find a labeled but unscanned vertex i and scan it as follows. For each arc (i, j), if $(i, j) \in F$ and j is unlabeled, give j the label (i^+). For each arc (j, i), if $(j, i) \in B$ and j is unlabeled, give j the label (i^-). Do not label any other neighbors of i. Vertex i has now been scanned.

Step 3.3. If vertex t has been labeled, go to Step 4. Otherwise, go to Step 3.1.

Step 4. Starting at vertex t, use the index labels to construct an augmenting chain. The label on vertex t indicates the next-to-last vertex in this chain, the label on that vertex indicates its predecessor in the chain, and so on. Stop and output this chain.

Step 5. Stop and output the message that the flow x is a maximum.

Let us illustrate this algorithm with the flow of Figure 13.10(b). We begin by labeling vertex s by $(-)$. Then we find that F consists of the arcs $(1, 4)$, $(2, 4)$, $(3, 6)$, $(3, 7)$, $(4, 5)$, $(5, 2)$, and $(6, 7)$, and B consists of all arcs except $(2, 4), (5, 2)$,

[12] This subsection may be omitted if time is short.

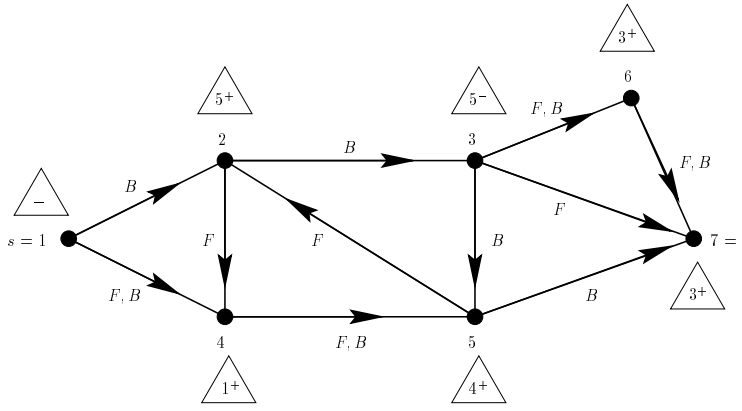


Figure 13.15: The labeling algorithm applied to the flow of Figure 13.10(b). The label is shown in a triangle next to a vertex. The arcs in F and in B are labeled F and B, respectively.

and $(3,7)$. We now go through the labeling and scanning procedure (Step 3). Figure 13.15 shows the labels. Note that vertex $s = 1$ is labeled but unscanned. The only arc $(1,x)$ in F is $(1,4)$. We therefore label vertex 4 with (1^+). There are no arcs $(x,1)$ in B. Thus, vertex 1 has been scanned. Since vertex $7 = t$ has not yet been labeled, we find another labeled, but unscanned vertex, namely 4. We now consider the arcs $(4,x)$ in F, namely the arc $(4,5)$. Thus, vertex 5 gets the label (4^+). We also note that no arc $(x,4)$ is in B. Thus, vertex 4 has been scanned. Note that t has not yet been labeled, so we find another labeled but unscanned vertex, namely 5. The arc $(5,2)$ is in F. Hence, we label vertex 2 with (5^+). The arc $(3,5)$ is in B, so we label vertex 3 with the label (5^-). Then vertex 5 has been scanned. Now t has not yet been labeled. We find a labeled but unscanned vertex. We have a choice of vertices 2 and 3. Suppose that we pick 3. Scanning 3 leads to the label (3^+) on vertices 6 and 7. Now $t = 7$ has been labeled. (Coincidentally, all vertices have been labeled.) We go to Step 4 and read backward to find a flow-augmenting chain. In particular, the label (3^+) on vertex 7 sends us back to vertex 3. The label (5^-) here sends us back to vertex 5, the label (4^+) to vertex 4, and the label (1^+) to vertex 1. Thus, we have the flow-augmenting chain $1,4,5,3,7$. In this example we did not need to differentiate a $+$ label from a $-$ label in order to find a flow-augmenting chain. However, had there been two arcs between 3 and 5, $(3,5)$ and $(5,3)$, the label 5^- on vertex 3 would have told us to use the backward arc $(3,5)$. The $+$ and $-$ labels will be useful in modifying the labeling algorithm to compute the number λ needed for the Max-Flow Algorithm.

Theorem 13.8 The labeling algorithm finds an augmenting chain if the flow x is not a maximum and ends by concluding that x is maximum otherwise.

Proof.[13] It is clear that if the algorithm produces a chain from s to t, the chain is

[13] The proof may be omitted.

augmenting. We show that if t has not yet been labeled and there is no labeled but unscanned vertex, then the flow is a maximum. Let S consist of all labeled vertices and T of all unlabeled vertices. Then (S,T) is an (s,t)-cut. Moreover, every arc from i in S to j in T is saturated and every arc from j in T to i in S has flow 0, for otherwise we could have labeled a vertex of T in scanning the vertices of S. We conclude that (S,T) is a saturated cut. By Corollary 13.5.1, we conclude that x is a maximum flow. Q.E.D.

In closing this subsection, we note that the labeling algorithm can be modified so that at the end, it is easy to compute the number λ needed for the Max-Flow Algorithm. At each step where we assign a label to a vertex j, we have just found an augmenting chain from s to j. We then let $\lambda(j)$ be the minimum of the numbers s_{uv} for (u,v) a forward arc of this chain and x_{uv} for (u,v) a backward arc of the chain. We let $\lambda(s)$ be $+\infty$. If we label j by scanning from i, we can compute $\lambda(j)$ from $\lambda(i)$. In particular, if j gets labeled (i^+), $\lambda(j) = \min\{\lambda(i), s_{ij}\}$, and if j gets labeled (i^-), then $\lambda(j) = \min\{\lambda(i), x_{ji}\}$. Finally, λ is $\lambda(t)$ (see Exercise 40). To illustrate, in our example, we would compute the following $\lambda(j)$ in the following order: $\lambda(s) = +\infty, \lambda(4) = \min\{\lambda(s), s_{14}\} = 2, \lambda(5) = \min\{\lambda(4), s_{45}\} = 1, \lambda(2) = \min\{\lambda(5), s_{52}\} = 1, \lambda(3) = \min\{\lambda(5), x_{35}\} = 1, \lambda(6) = \min\{\lambda(3), s_{36}\} = 1, \lambda(7) = \min\{\lambda(3), s_{37}\} = 1$, and conclude that $\lambda = \lambda(7) = 1$.

13.3.7 Complexity of the Max-Flow Algorithm

Let us make a comment on the computational complexity of the Max-Flow Algorithm. Let a be the number of arcs in the directed network and v be the maximum flow. The labeling algorithm described in Section 13.3.6 requires at most $2a$ arc inspections in each use to find an augmenting chain—we look at each arc at most once as a forward arc and at most once as a backward arc. If all capacities are integers and we start the Max-Flow Algorithm with the flow $x_{ij} = 0$ for all i, j, then each flow-augmenting chain increases the value by at least 1, so the number of iterations involving a search for an augmenting chain is at most v. Hence, the number of steps is at most $2av$.[14] One problem with this computation is that we want to determine complexity solely as a function of the size of the input, not in terms of the solution v. Note that our computation implies that the algorithm might take a long time. Indeed, it can. Consider the directed network of Figure 13.16. Starting with the 0 flow, we could conceivably choose first the augmenting chain $1,2,3,4$, then the augmenting chain $1,3,2,4$, then $1,2,3,4$, then $1,3,2,4$, and so on. This would require 2 billion iterations before converging! However, if we happen to choose first the augmenting chain $1,2,4$, and then the augmenting chain $1,3,4$, then

[14] This disregards the simple computations of computing slacks and modifying the flow. These are each applied to each arc at most twice in each iteration, so add a constant times av to the number of steps. In each iteration, we also have to compute λ, which is the minimum of a set of no more than a numbers. The total number of computations in computing all the λ's is thus again at most av. Finally, we have to construct the augmenting chain from the labels, which again takes at most a steps in each iteration, or at most av steps in all. In sum, if k is a constant, the total number of steps is thus at most kav, which is $O(av)$, to use the notation of Section 2.18.

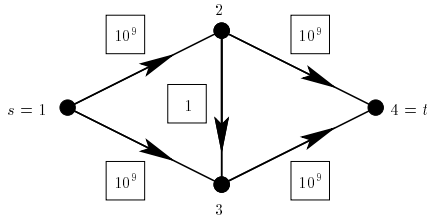


Figure 13.16: A poor choice of augmenting chains leads to 2 billion iterations in the Max-Flow Algorithm.

we finish in two iterations! To avoid these sorts of problems, we change Step 3.2 of the labeling algorithm so that vertices are scanned in the same order in which they receive labels. Edmonds and Karp [1972] have shown that in this case, a maximum flow is obtained in at most $an/2$ applications of the labeling algorithm, where n is the number of vertices. Thus, since each use of the labeling algorithm requires at most $2a$ steps, the total number of steps[15] is at most

$$(2a)(an/2) = a^2 n \leq [n(n-1)]^2 n.$$

The Edmonds and Karp algorithm has since been improved. See Ahuja, Magnanti, and Orlin [1993] and Papadimitriou and Steiglitz [1982] for references on improved algorithms.

13.3.8 Matching Revisited[16]

In this subsection we investigate the relation between network flows and the matchings we studied in Chapter 12. In particular, we show how to prove Theorem 12.5, the result that in a bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum covering.

Suppose that $G = (X, Y, E)$ is a bipartite graph. We can make it into a directed network D, called the *associated network*, by adding a source s, a sink t, arcs from s to all vertices of X and from all vertices of Y to t and by directing all edges between X and Y from X to Y. We put capacity 1 on all new arcs, and capacity ∞ (or a very large number) on all arcs from X to Y. Then it is easy to see that any integer flow (a flow all of whose x_{ij} values are integers) from s to t in D corresponds to a matching M in G: We include an edge $\{i, j\}$ in M if and only if there is positive flow along the arc (i, j), $i \in X$, $j \in Y$. Conversely, any matching M in G defines an integer flow x in D: For $\{i, j\}$ in E with i in X, take x_{ij} to be 1 if $\{i, j\}$ is in M, and 0 otherwise; take x_{sk} to be 1 if and only if k is saturated in M, and x_{lt} to be 1 if and only if l is saturated in M. Moreover, the number of edges in the matching M equals the value of the flow x. Figure 13.17 illustrates this construction by showing a bipartite graph G and the associated network D. The flow shown in D corresponds to the matching shown by wiggly edges in G. We summarize these results in the next theorem.

[15] As per the previous footnote (14), a more accurate estimate is at most $ka(an/2)$ steps.
[16] This subsection may be omitted.

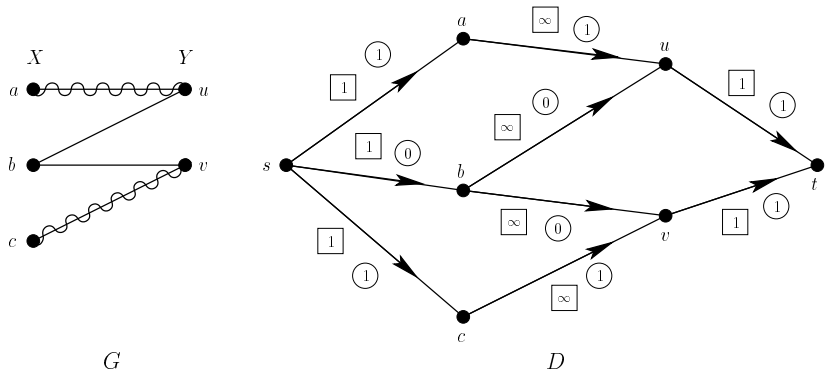


Figure 13.17: A bipartite graph G and its associated network D. The flow in D shown by encircled numbers corresponds to the matching in G shown by wiggly edges. Capacities are shown in squares.

Theorem 13.9 Suppose that $G = (X, Y, E)$ is a bipartite graph and D is the associated network. Then there is a one-to-one correspondence between integer (s,t)-flows in D and matchings in G. Moreover, the value of an integer flow is the same as the number of edges in the corresponding matching.

Since matchings in a bipartite graph $G = (X, Y, E)$ correspond to flows in the associated network D, we might ask what coverings in the graph correspond to in the network. The answer is that coverings correspond to (s,t)-cuts of finite capacity. To make this precise, suppose that $A \subseteqq X$ and $B \subseteqq Y$. Any covering K in G can be written in the form $A \cup B$ for such A and B. Any (s,t)-cut (S,T) of D can be written in the following form:

$$S = \{s\} \cup (X - A) \cup B, \quad T = \{t\} \cup A \cup (Y - B);$$

for s is in S and t is in T. We simply define A to be all vertices in $T \cap X$ and B to be all vertices in $S \cap Y$ (see Figure 13.18). We now have the following theorem.

Theorem 13.10 Suppose that $G = (X, Y, E)$ is a bipartite graph and D is the associated network. Suppose that $A \subseteqq X$ and $B \subseteqq Y$. Let $K = A \cup B$ and let

$$S = \{s\} \cup (X - A) \cup B, \quad T = \{t\} \cup A \cup (Y - B). \tag{13.12}$$

Then K is a covering of G if and only if (S,T) is an (s,t)-cut of D of finite capacity. Moreover, if (S,T) is an (s,t)-cut of finite capacity, $|A \cup B|$ is the capacity of the cut (S,T).

Proof. Suppose that (S,T) defined by (13.12) is an (s,t)-cut of finite capacity. Since the cut has finite capacity, there can be no arcs from X to Y in the cut, that is, no arcs from $X - A$ to $Y - B$ (see Figure 13.18). Thus, by (13.12), all arcs from X to Y in D go from A or to B, so all edges of G are joined to A or to B, so

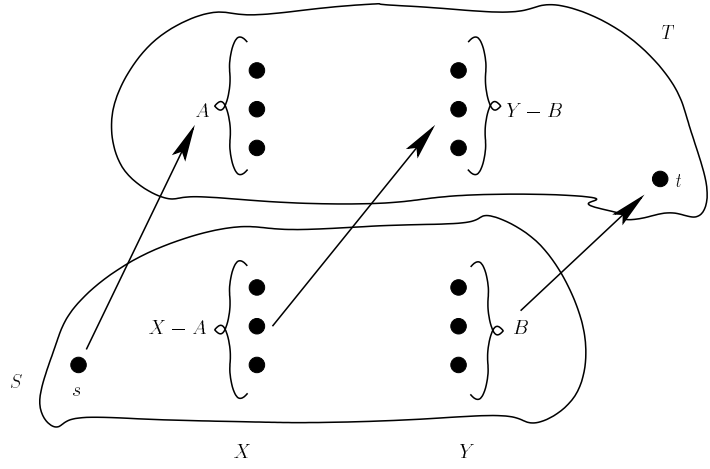


Figure 13.18: The cut (13.12). The only possible arcs in the cut are shown by arrows.

$K = A \cup B$ is a covering. Next, note that $|A \cup B|$ is the capacity of the cut (S, T). For by Figure 13.18, the arcs in the cut are exactly the arcs (s, x) for x in A and (y, t) for y in B. There are $|A \cup B|$ such arcs. Each has unit capacity, so the cut has capacity $|A \cup B|$.

It remains to prove that if $K = A \cup B$ is a covering of G, then (S, T) defined by (13.12) gives an (s, t)-cut and it has finite capacity. This is left to the reader (Exercise 37). Q.E.D.

We can now prove Theorem 12.5.

Proof of Theorem 12.5. [Theorem 12.5: Suppose that $G = (X, Y, E)$ is a bipartite graph. Then the number of edges in a maximum matching equals the number of vertices in a minimum covering.]

Consider the associated network. Since all the capacities are integers, it follows from the proof of Theorem 13.7 that there is a maximum flow in which all x_{ij} are integers. The value of this flow will also have to be an integer, and will, by Theorem 13.9, give the number α of edges in the maximum matching. Moreover, by the Max-Flow Min-Cut Theorem (Theorem 13.6), this number α is also the minimum capacity of an (s, t)-cut (S, T). Suppose that this cut is given by (13.12). Now clearly, an (s, t)-cut of minimum capacity has finite capacity, so by Theorem 13.10, (S, T) corresponds via (13.12) to a covering $K = A \cup B$. The number of vertices in K is equal to $|A \cup B|$, which is equal to the capacity of the cut (S, T), which is equal to α. Then we have a covering of α vertices and a matching of α edges, so by Theorem 12.4, the covering must be a minimum. Hence, the number of edges in a maximum matching equals the number of vertices in a minimum covering. Q.E.D.

13.3.9 Menger's Theorems[17]

The Max-Flow Min-Cut Theorem has useful application in the design of communication networks. In designing such networks, we like to make sure that their "connectivity" remains intact after some connections are destroyed and we are therefore interested in the minimum number of arcs whose removal destroys all paths from a vertex a to a vertex z. One way to preserve connectivity is to make sure that we build in "redundancy" and have a lot of paths between pairs of vertices. Let us say that a pair of paths from a to z is *arc-disjoint* if they do not have any arcs in common. We shall see that there is a close relationship between the minimum number of arcs whose removal destroys all (simple) paths from a to z and the maximum number of arc-disjoint simple paths from a to z. The result is one of a series of remarkable theorems known as Menger's Theorems (after Menger [1927]).

Theorem 13.11 Let N be a connected directed network with source a and sink z in which each arc has capacity 1. Then:

(a) The value of a maximum (a, z)-flow in N is equal to the maximum number M of arc-disjoint simple paths from a to z.

(b) The capacity of a minimum (a, z)-cut in N is equal to the minimum number p of arcs whose removal destroys all simple paths from a to z.

Proof. We give the proof in the case where a has only outgoing arcs and z only incoming arcs. The proof in the general case is left to the reader (Exercise 32).

(a) Let F be the set of all integers k such that if there is an (a, z)-flow of value k in a directed network with capacities all 1, there is a set of k arc-disjoint simple paths from a to z. Clearly, $k = 1$ is in F. We now argue by induction on k that all integers are in F. We assume that k is in F and show that $k + 1$ is in F. Suppose that there is a flow of value $k + 1$. Then there is a simple path P from a to z each arc of which has nonzero flow. Since the capacities are all 1, the flows are 1 on each arc of P. The remaining k units of flow must go through arcs not on P. Thus, deleting the arcs of P leaves a directed network with all capacities 1 and with a flow of value k. By the inductive hypothesis, there is a set of k arc-disjoint simple paths from a to z in this new directed network. These paths together with P form a set of $k + 1$ arc-disjoint simple paths from a to z in the original directed network. This shows that if v is the value of a maximum (a, z)-flow, then $v \leq M$.

Suppose that we have M arc-disjoint simple paths from a to z in N. If we define x_{ij} to be 1 on all arcs (i, j) that are in such a path and 0 otherwise, then clearly $x = \{x_{ij}\}$ is a flow with value M and so $v \geq M$.

(b) Let (S, T) be an (a, z)-cut in N. Then if all arcs from S to T are destroyed, z is not reachable from a. It follows that the capacity of this cut is at least p.

Now let A be a set of p arcs whose deletion destroys all simple paths from a to z, and let N' be the directed network obtained from N by removing arcs of A. Let B be the set of all vertices reachable from a in N'. By definition of B, there can

[17] This subsection may be omitted.

be no arc from a vertex of B to a vertex of $B' = V(N) - B$, so all arcs from B to B' in N are in A. It follows that (B, B') is a cut of capacity at most p and thus it must be a minimum cut. Q.E.D.

Theorem 13.12 (Menger [1927]) The maximum number of arc-disjoint simple paths from a vertex a to a vertex z in a digraph D is equal to the minimum number of arcs whose deletion destroys all simple paths from a to z in D.

Proof. Build a directed network with source a and sink z by assigning unit capacity to each arc of D. The result follows from Theorem 13.11 and the Max-Flow Min-Cut Theorem (Theorem 13.6). Q.E.D.

To illustrate this theorem, consider the directed network of Figure 13.10 (disregarding capacities). Then there are two arc-disjoint simple paths from s to t: namely, $s, 2, 3, 6, t$ and $s, 4, 5, t$. There are also two arcs whose deletion destroys all (simple) paths from s to t: arcs $(2, 3)$ and $(4, 5)$. Thus, it follows that the maximum number of arc-disjoint simple paths from s to t is 2 and the minimum number of arcs whose deletion destroys all (simple) paths from s to t is also 2.

We say that two paths from a to z in a digraph are *vertex-disjoint* if they do not have a vertex in common except a and z.

Theorem 13.13 (Menger [1927]) Suppose that there is no arc from vertex a to vertex z in digraph D. Then the maximum number of vertex-disjoint simple paths from a to z in D is equal to the minimum number of vertices whose removal from D destroys all simple paths from a to z.

Proof. Define a new digraph D' by splitting each vertex u other than a and z into two new vertices u_1 and u_2, adding an arc (u_1, u_2), replacing each arc (u, w) with arc (u_2, w_1), replacing each arc (u, z) with (u_2, z), and replacing each arc (a, u) with (a, u_1). Then one can show that the maximum number of arc-disjoint simple paths from a to z in D' is equal to the maximum number of vertex-disjoint simple paths from a to z in D and the minimum number of arcs in D' whose deletion destroys all simple paths from a to z is equal to the minimum number of vertices whose removal destroys all simple paths from a to z in D. Details are left to the exercises [Exercises 33(a) and (b)]. Q.E.D.

To illustrate this theorem, consider again the directed network of Figure 13.10. The two arc-disjoint simple paths from s to t given above are also vertex-disjoint. Two vertices whose removal destroys all simple paths from s to t are vertices 2 and 4.

EXERCISES FOR SECTION 13.3

1. In each directed network of Figure 13.19, a (potential) (s, t)-flow is shown in the circled numbers.

   (a) Which of these flows is feasible? Why?

   (b) If the flow is feasible, find its value.

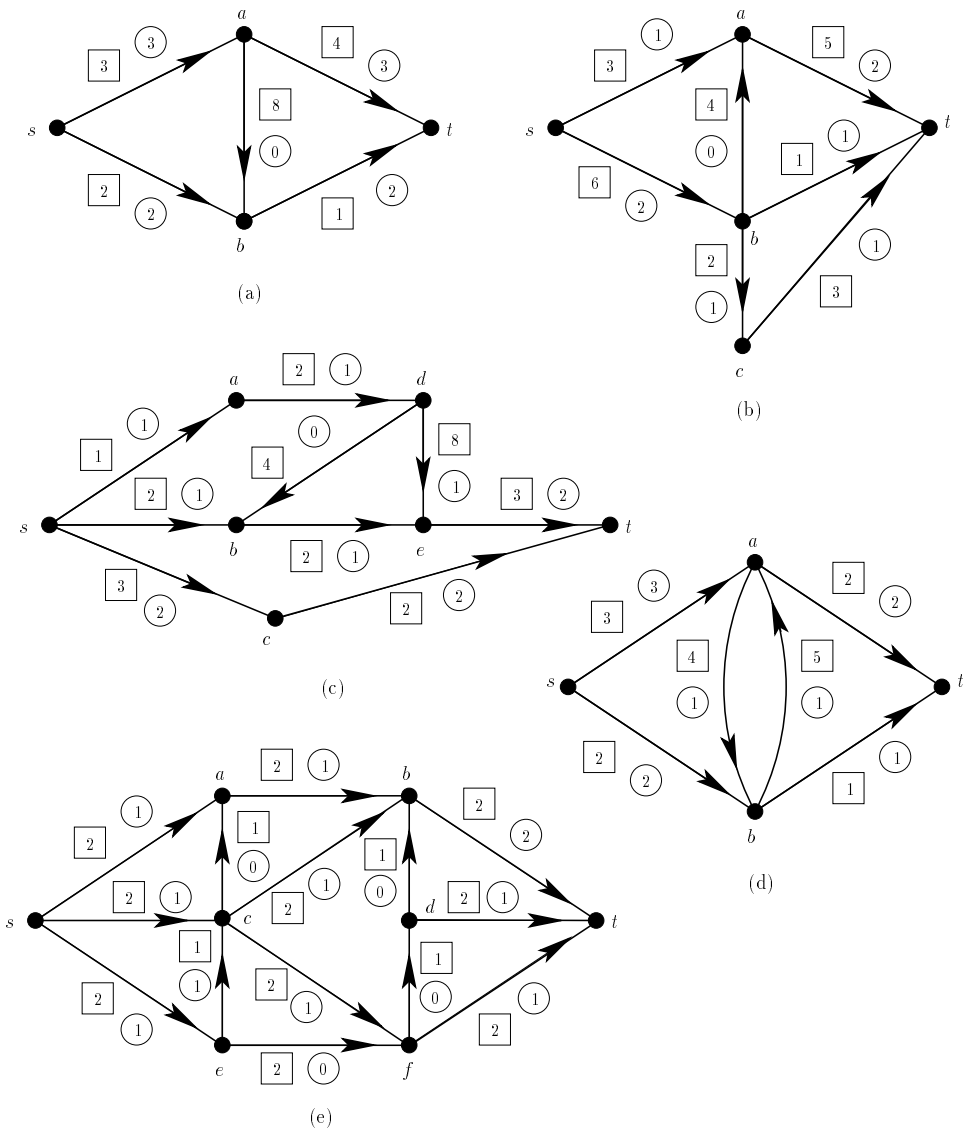


Figure 13.19: (Potential) flows for exercises of Section 13.3.

2. In each directed network D of Figure 13.8, interpret the weights as capacities and find the capacity of the cut (S, T) where $S = \{a, b, e\}$ and $T = V(D) - S$.
3. In each flow x of Figure 13.20, compute the slack on each arc.
4. In the flow III of Figure 13.13, which of the following are flow-augmenting chains?

   (a) $1, 2, 3, 7$ (b) $1, 2, 5, 7$ (c) $1, 4, 5, 7$ (d) $1, 4, 2, 3, 7$

5. In the flow III of Figure 13.12, which of the chains of Exercise 4 are flow-augmenting chains?
6. For each flow of Figure 13.20, either show that it is maximum by finding an (s, t)-cut (S, T) such that $v = c(S, T)$ or show that it is not maximum by finding an augmenting chain.
7. Give an example of a directed network and a flow x for this network which has value 0 but such that x_{ij} is not 0 for all i, j.
8. Illustrate Menger's (first) Theorem, Theorem 13.12, on the directed networks of:

   (a) Figure 13.19 (b) Figure 13.21

9. Illustrate Menger's (second) Theorem, Theorem 13.13, on the directed networks of:

   (a) Figure 13.19 (b) Figure 13.21

10. Let G be an undirected graph.

    (a) State and prove a variant of Menger's (first) Theorem, Theorem 13.12.

    (b) State and prove a variant of Menger's (second) Theorem, Theorem 13.13.

11. Apply Algorithm 13.4 to each directed network of Figure 13.8 if weights are interpreted as capacities and $s = a$, $t = z$.
12. Repeat Exercise 11 using Algorithm 13.5.
13. For each flow of Figure 13.20, apply Algorithm 13.6 to search for an augmenting chain.
14. In each application of Algorithm 13.6 in Exercise 13, also calculate the numbers $\lambda(j)$.
15. In directed network (d) of Figure 13.8, let $s = a$ and $t = z$ and define a flow by letting $x_{ab} = x_{bc} = x_{cd} = x_{dz} = 2$ and $x_{ae} = x_{ef} = x_{fg} = x_{gz} = 3$, and $x_{ah} = x_{hi} = x_{ij} = x_{jz} = 2$, and otherwise taking $x_{ij} = 0$. Apply Algorithm 13.6 to search for a flow-augmenting chain.
16. Let G be a graph and S and T be disjoint subsets of the vertices. Show that the maximum number of vertex-disjoint simple paths with one end in S and one end in T is equal to the minimum number of vertices whose deletion separates S from T in the sense that after deletion, no connected component contains both a vertex of S and a vertex of T.
17. For each bipartite graph G of Figure 12.12:

    (a) Find the associated network D.

    (b) Find an integer (s, t)-flow in D and its corresponding matching in G.

    (c) Find an (s, t)-cut in D and its corresponding covering in G.

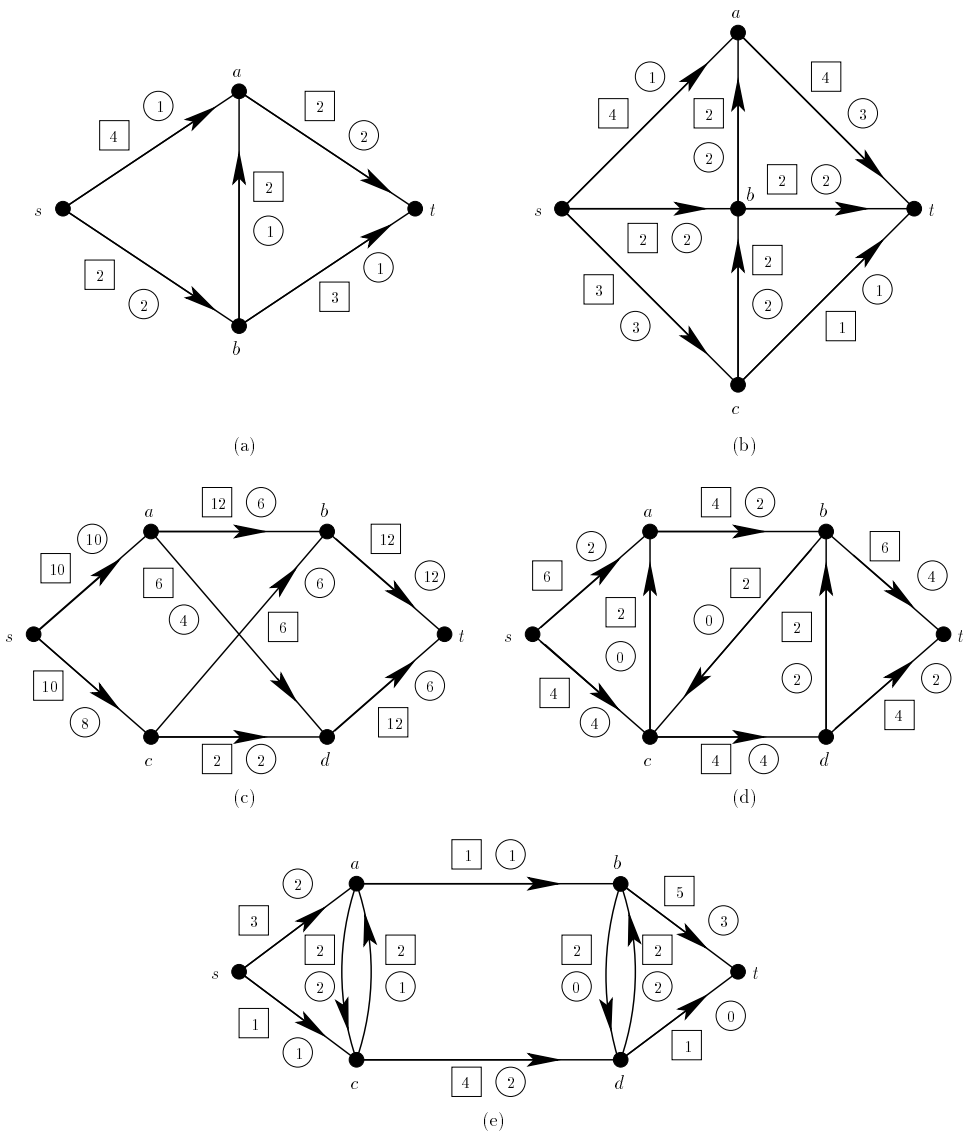


Figure 13.20: Flows for exercises of Sections 13.3 and 13.4.

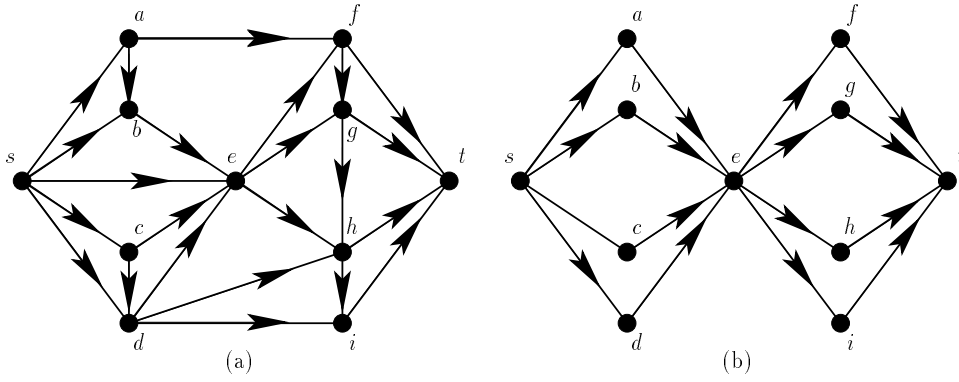


Figure 13.21: Digraphs for exercises of Section 13.3.

18. Consider the college admissions problem of Example 13.8. The following chart contains the list of 8 applicants, including their potential major and their home state.

	Potential Major				
Applicant	Chem.	Bio.	Math.	Geo.	Home State
A	x				MD
B		x			MD
C			x	x	NY
D	x	x			MD
E			x	x	PA
F			x		NJ
G			x		NY
H				x	PA

(a) Draw the associated directed network.

(b) If the college wants at most 2 students from each state, can the admissions staff do their job?

(c) If the college wants at most 1 student from each state, can the admissions staff do their job?

19. Consider the distributed computing on a two-processor computer problem of Example 13.9. Table 13.4 gives the execution costs for each module on the two processors and Table 13.5 gives the interprocessor communication costs.

(a) Draw the associated network.

(b) Find the minimum-cost assignment of modules to processors using a minimum-cost cut.

20. Recall that a flow from a source s to a sink t in an undirected network is defined as a flow from s to t in the directed network obtained by replacing each edge $\{i, j\}$ by the two arcs (i, j) and (j, i) and letting each arc have the same capacity as the corresponding edge. Find a maximum flow from s to t in each network of Figure 13.22.

Table 13.4: Execution Costs

i	1	2	3	4	5
a_i	6	5	10	4	8
b_i	3	6	5	10	4

Table 13.5: Interprocessor Communication Costs

	1	2	3	4	5
1	0	4	1	0	0
2	4	0	5	0	0
3	1	5	0	6	2
4	0	0	6	0	1
5	0	0	2	1	0

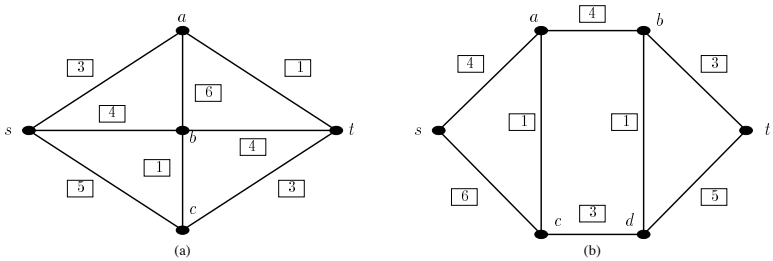


Figure 13.22: Networks for exercises of Section 13.3.

21. A pipeline network sends oil from location A to location B. The oil can go via the northern route or the southern route. Each route has one junction, with a pipeline going from the junction on the southern route to the junction on the northern route. The first leg of the northern route, from location A to the junction, has a capacity of 400 barrels an hour; the second leg, from the junction to location B, has a capacity of 300 barrels an hour. The first leg of the southern route has a 500-barrel/hour capacity, and the second leg has a 300-barrel hour capacity. The pipeline joining the two junctions also has a 300-barrel/hour capacity. What is the largest number of barrels of oil that can be shipped from location A to location B in an hour?

22. In this exercise we build on the notion of reliability of systems discussed in Example 2.21 and in Example 3.10. Suppose that a system is represented by a directed network, with the components corresponding to arcs. Let us say that the system works if and only if in the modified network defined by working components, there is an (s,t)-flow of value at least v. For each directed network of Figure 13.23, compute $F(x_1x_2\cdots x_n)$ as defined in Example 2.21 if v is 3.

23. Suppose that D is a directed network and X is a set of sources and Y is a set of sinks. [We assume that $X, Y \subseteq V(D)$ and $X \cap Y = \emptyset$.] An (X,Y)-*flow* is a flow where the conservation conditions (13.6) hold only for vertices that are not sources or sinks. The *value* of the flow is defined to be

$$\sum_{\substack{i\in X\\ j\notin X}} x_{ij} - \sum_{\substack{i\in X\\ j\notin X}} x_{ji}.$$

We can find a maximum (X,Y)-flow by joining two new vertices, s and t, to D,

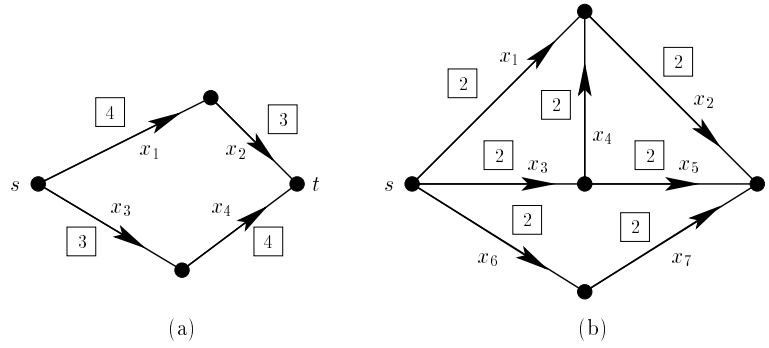


Figure 13.23: Directed networks for exercises of Section 13.3.

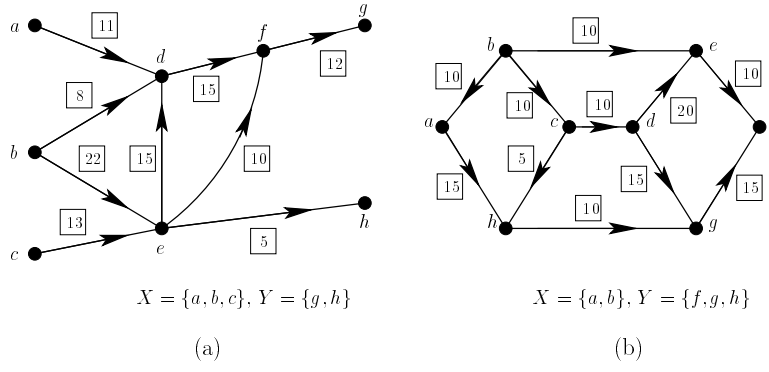


Figure 13.24: Directed networks with a set X of sources and a set Y of sinks.

adding arcs of capacity ∞ from s to all vertices in X and from all vertices in Y to t, and finding a maximum (s, t)-flow in the new network. Find a maximum (X, Y)-flow in each directed network of Figure 13.24.

24. There are three warehouses, w_1, w_2, and w_3, and three retail outlets, r_1, r_2, and r_3. The warehouses have, respectively, 3000, 4000, and 6000 drums of paint, and the retail outlets have demand for, respectively, 2000, 4000, and 3000 drums. Figure 13.25 shows a freight network, with the capacity on arc (i, j) giving the largest number of drums that can be shipped from location i to location j during a given day. Can all the demands be met if only one day is allowed for shipping from warehouses? (A drum can go along as many arcs as necessary in one day.) If not, what is the largest total demand that can be met? Solve this problem by translating it into a multisource, multisink problem and then into an ordinary network flow problem (see Exercise 23). This is an example of a *transshipment problem*.

25. (Ahuja, Magnanti, and Orlin [1993]) A union has a certain number of skilled craftsmen with varied skills and varying levels of seniority. Each person has at least one of a set of designated skills but only one level of seniority. The union wants to organize a governing board subject to certain conditions: One person with each type of skill is a member of the governing board, and the number of people with seniority level k

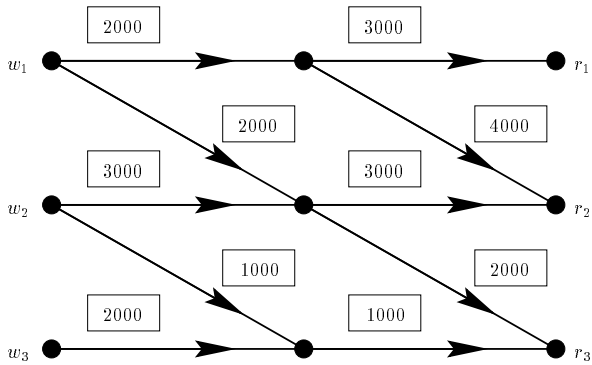


Figure 13.25: Freight network for Exercise 24, Section 13.3.

is at most u_k. Is it possible to find such a governing board? Show that the answer is "yes" if and only if a certain (s, t) directed network has a maximum-flow value equal to the number of designated skills.

26. Several legislative committees with no members in common are gathering to exchange ideas. We would like to assign them to different rooms so that no two people on the same committee go to the same room. Show how to assign committee members to rooms using a maximum-flow formulation. You should use the number of people in each committee and the capacity of the ith meeting room as input to your formulation.

27. In some directed networks, we might have vertex capacities as well as arc capacities. For instance, there might be limited capacity for the number of cars that can pass through a given toll facility. Then we seek a maximum flow satisfying both arc and vertex capacity constraints. Transform this problem into a standard maximum-flow problem with only arc capacity constraints.

28. Show that if x is an (s, t)-flow of value v and (S, T) is an (s, t)-cut of capacity c, then $v = c$ if and only if for each arc (i, j) from T to S, $x_{ij} = 0$, and for each arc (i, j) from S to T, $x_{ij} = c_{ij}$.

29. Suppose that a flow from s to t is decomposed into unit flow paths from s to t, and each of these unit flow paths crosses a given saturated cut exactly once. Show that the flow is maximum.

30. We wish to send messengers from a location s to a location t in a region whose road network is modeled by a digraph. Because some roads may be blocked, we wish each messenger to drive along a route that is totally disjoint from that of all other messengers. How would we find the largest number of messengers who could be sent? (*Hint:* Use unit capacities.)

31. In Example 13.8, prove:

    (a) If a maximum-flow value equals the number of majors represented among the applicants' interests, it solves the "college admissions" problem.

    (b) If not, there is no solution.

32. Complete the proof of Theorem 13.11 in the case that there may be incoming arcs to vertex a or outgoing arcs from vertex z.

33. This exercise establishes the proof of Theorem 13.13. Suppose that there is no arc from vertex a to vertex z in digraph D. Define D' as in the "proof" of the theorem.

    (a) Show that the maximum number of arc-disjoint simple paths from a to z in D' is equal to the maximum number of vertex-disjoint simple paths from a to z in D.

    (b) Show that the minimum number of arcs in D' whose deletion destroys all (simple) paths from a to z is equal to the minimum number of vertices whose removal destroys all simple paths from a to z in D.

34. Let N be an undirected network whose underlying graph is connected. Recall that a *cut* in N is a partition of the vertices into two sets S and T or, equivalently, the set of all edges joining a vertex of S to a vertex of T. If s and t are the source and sink of N, respectively, (S, T) is an (s, t)-*cut* if $s \in S$ and $t \in T$. Let F be a set of edges of N.

    (a) Show that if F is a simple cut set in the sense of Exercise 18, Section 13.1, it is a cut.

    (b) Show that if F is an (s, t)-cut with a minimum capacity, F is a simple cut set.

35. Suppose that C is a flow-augmenting chain and λ is the capacity of C as defined in Section 13.3.4. If we increase each x_{ij} on a forward arc by λ and decrease each x_{ij} on a backward arc by λ, show that even if C is not a simple chain, the value or net flow out of s is increased.

36. Show that in the Max-Flow Algorithm, it suffices to find flow-augmenting chains that are simple chains.

37. Complete the proof of Theorem 13.10.

38. Use the results of Section 13.3.8 to prove the König-Egerváry Theorem (Corollary 12.5.1) without first proving Theorem 12.5.

39. Suppose that $G = (X, Y, E)$ is a bipartite graph and M is a matching of G. Use the results of Section 13.3.8 to prove Theorem 12.7, namely, that M is maximum if and only if G contains no M-augmenting chain.

40. Show that in the labeling algorithm, if $\lambda(j)$ is defined as in the discussion following the proof of Theorem 13.8, then $\lambda = \lambda(t)$.

13.4 MINIMUM-COST FLOW PROBLEMS

13.4.1 Some Examples

An alternative network flow problem, which has many important special cases, is the following. Suppose that in a directed network, in addition to a nonnegative capacity c_{ij} on each arc, we have a nonnegative cost a_{ij} of shipping one unit of flow from i to j. We want to find a flow sending a fixed nonnegative number v of units from source s to sink t, and doing so at minimum cost. That is, we want to find

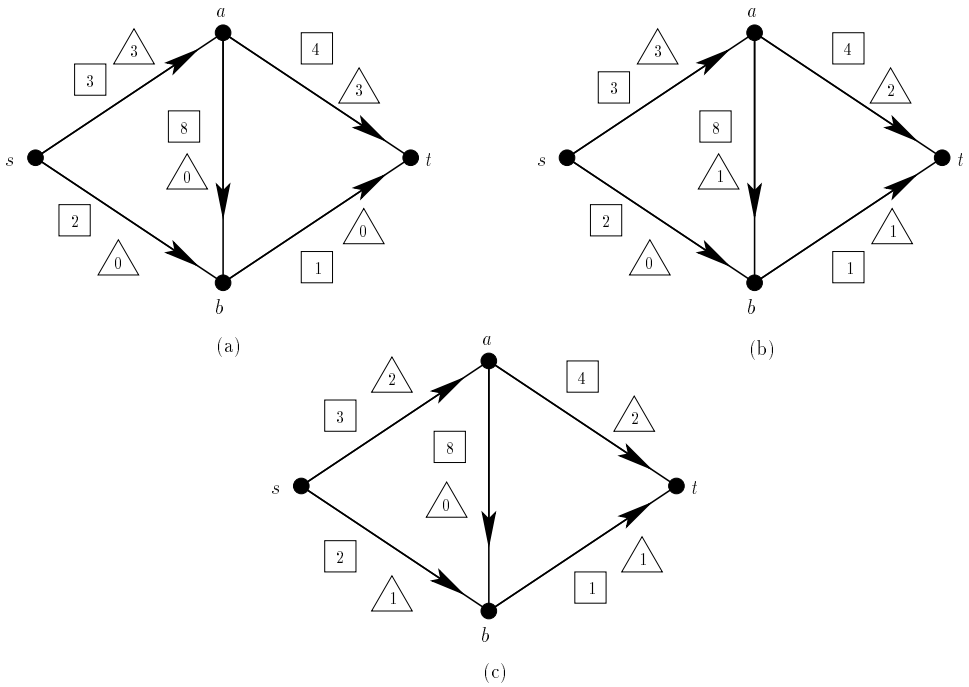


Figure 13.26: The three flows of value 3 for the directed network (a) of Figure 13.19. The capacity is shown in a square, the flow in a triangle.

a flow $x = \{x_{ij}\}$ such that its value is v and such that $\sum a_{ij}x_{ij}$ is minimized. We call this a *minimum-cost flow problem.*

To illustrate, consider the directed network (a) of Figure 13.19. Consider the number in the square as the capacity and the number in the circle as the cost. There are three flows that attain a value of 3; these are shown in Figure 13.26. Of these flows, flow (c) has the minimum cost: namely,

$$2 \cdot 3 + 2 \cdot 3 + 0 \cdot 0 + 1 \cdot 2 + 1 \cdot 2 = 16.$$

Flows (a) and (b) have values 18 and 17, respectively.

Example 13.10 Routing a Traveling Party (Minieka [1978]) A traveling party of 75 people is to go from New York to Honolulu. What is the least-cost way to route the party? The solution is obtained by finding a minimum-cost flow of 75 from New York to Honolulu in a directed network where the vertices are cities, an arc indicates a direct air link, and there is a capacity constraint (unbooked seats) and a cost constraint (air fare) on each arc. (This assumes that air fares are simple sums along links, which usually would not be true.) ■

Example 13.11 The Transportation Problem Imagine that a particular commodity is stored in n warehouses and is to be shipped to m markets. Let a_i be the supply of the commodity at the ith warehouse, let b_j be the demand for the commodity at the jth market, and let a_{ij} be the cost of transporting one unit of the commodity from warehouse i to market j. For simplicity, we assume that $\sum a_i = \sum b_j$, that is, that the total supply equals the total demand. (This assumption can easily be eliminated—see Exercise 10.) The problem is to find a shipping pattern that minimizes the total transportation cost.

This problem can be formulated as follows. Let x_{ij} be the number of units of the commodity shipped from i to j. We seek to minimize

$$\sum_{\substack{i=1 \\ j=1}}^{\substack{m \\ n}} a_{ij}x_{ij}$$

subject to the following constraints: For every i,

$$\sum_{j=1}^{m} x_{ij} \leq a_i, \tag{13.13}$$

and for every j,

$$\sum_{i=1}^{n} x_{ij} \geq b_j. \tag{13.14}$$

Constraint (13.13) says that the total amount of commodity shipped from the ith warehouse is at most the amount there, and constraint (13.14) says that the total amount of commodity shipped to the jth market is at least the amount demanded. Note that since $\sum a_i = \sum b_j$, any solution satisfying (13.13) and (13.14) for all i and j will also satisfy

$$\sum_{j=1}^{m} x_{ij} = a_i \tag{13.15}$$

and

$$\sum_{i=1}^{n} x_{ij} = b_j. \tag{13.16}$$

We can look at this transportation problem as a minimum-cost flow problem. Draw a digraph with vertices the n warehouses $w_1, w_2, \ldots, w_n$ and the m markets $k_1, k_2, \ldots, k_m$. Add a source vertex s and a sink vertex t, and include arcs from each warehouse to each market, from the source to all warehouses, and from each market to the sink (see Figure 13.27). On arc (w_i, k_j), place a cost a_{ij} and a capacity $c_{ij} = \infty$ (or a very large number); on arc (s, w_i), place a cost of 0 and a capacity of a_i; and on arc (k_j, t), place a cost of 0 and a capacity of b_j. Because we have constraints (13.13) and (13.14), it is easy to see that we have a minimum-cost flow problem for this directed network: We seek a flow of value equal to $\sum a_i = \sum b_j$, at minimum cost. ■

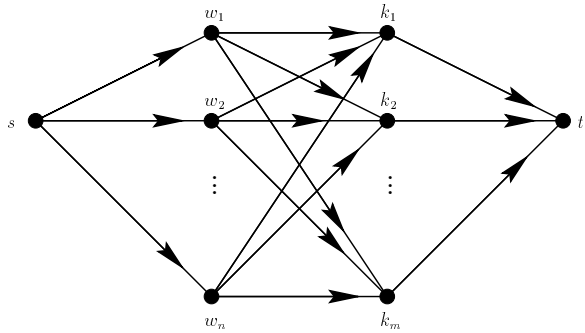


Figure 13.27: A directed network for the transportation problem.

Example 13.12 The Optimal Assignment Problem (Example 12.6 Revisited) In Example 12.6 and Section 12.7.2 we discussed the following job assignment problem. There are n workers and m jobs, every worker is suited for every job, and worker i's potential performance on job j is given a rating r_{ij}. We wish to assign workers to jobs so as to maximize the sum of the performance ratings. Let x_{ij} be a variable that is 1 if worker i is assigned to job j and 0 otherwise. Then we want to maximize

$$\sum_{\substack{i=1 \\ j=1}}^{\substack{m \\ n}} r_{ij} x_{ij}$$

under the constraints

$$\sum_{i=1}^{n} x_{ij} \leq 1 \quad \text{and} \quad \sum_{j=1}^{m} x_{ij} \geq 1. \tag{13.17}$$

(Note that in Section 12.7.2, we took $m = n$. Here we consider the general case.) We may think of this as a transportation problem (Example 13.11) by letting the workers correspond to the warehouses and the jobs to the markets. We then take all a_i and b_j equal to 1 and let the cost be $c_{ij} = -r_{ij}$. We seek to maximize $\sum_{i,j} r_{ij} x_{ij}$ or minimize $\sum_{i,j} c_{ij} x_{ij}$. Hence, this optimal assignment problem is also a minimum-cost flow problem. [One difference is that we have the additional requirement that $x_{ij} = 0$ or 1. However, it is possible to show that if a minimum-cost flow problem has integer capacities, costs, and value v, some optimal solution x is in integers, and standard algorithms produce integer solutions (see below). Then the requirement (13.17) and the added requirement $x_{ij} \geq 0$ are sufficient to give us $x_{ij} = 0$ or 1.] ■

Example 13.13 Building Evacuation (Ahuja, Magnanti, and Orlin [1993], Chalmet, Francis, and Saunders [1982]) The September 11, 2001 World Trade Center collapse has focused attention on methods for evacuation of tall buildings. Design of such buildings must allow for rapid evacuation. Models of building evacuation can help in the building design phase. The following is a simplified

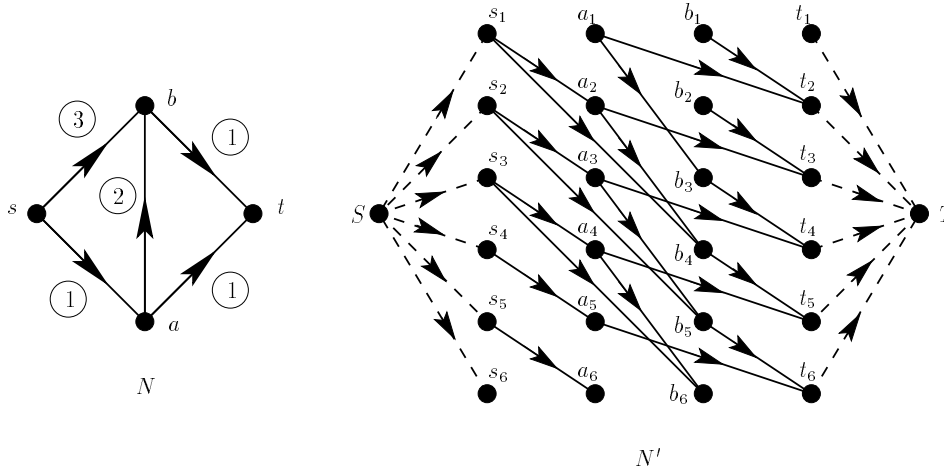


Figure 13.28: Directed networks N and N' for a hypothetical small building with $C = 6$. All capacities are 2 except for dashed arcs, which have infinite capacity (and zero cost). The numbers in the circles in N are times t_{ij}. The cost on arc (i_p, j_q) in N' is the cost on arc (i, j) in N.

model of the problem. There are many locations in a building from which people will have to be evacuated and many exits. For simplicity, we assume that there is only one location from which we need to evacuate people and only one exit, and these will become the source and sink, respectively, of a directed network N. We assume that we have a certain number v of people whom we need to evacuate to the exit. A building has various locations that will become the remaining vertices of the directed network N. We join vertex i to vertex j by an arc if there is a direct passage (stairway, hallway, etc.) from location i to location j and we let the capacity c_{ij} of this arc (i, j) be the number of people who can pass through this passage per unit time. Note that it might take several units of time t_{ij} to pass through a passage (i, j). We assume, again by way of simplifying the problem, that t_{ij} is an integer. We now replace the directed network N by a larger one, N', in which we make C copies of each of the vertices (including the source and sink), where C is chosen large enough to be sure that we can evacuate the building in p units of time. We think of copy i_p of vertex i as representing that vertex at time p. In the larger directed network N', we draw an arc from i_p to j_q if (i, j) is in N and $q - p = t_{ij}$. In other words, the arc (i_p, j_q) represents movement from vertex i to vertex j in the time it takes a person to move along the passage from i to j. Put capacity c_{ij} and "cost" t_{ij} on the new arcs (i_p, j_q). Add a source vertex S and sink vertex T and arcs (S, s_p) for all p and (t_q, T) for all q, with infinite capacities and zero cost. Figure 13.28 shows the directed networks N and N' for a hypothetical small building. A minimum "cost" flow of value v from S to T will give us a plan for evacuating all v persons from the building in a minimum amount of time. ■

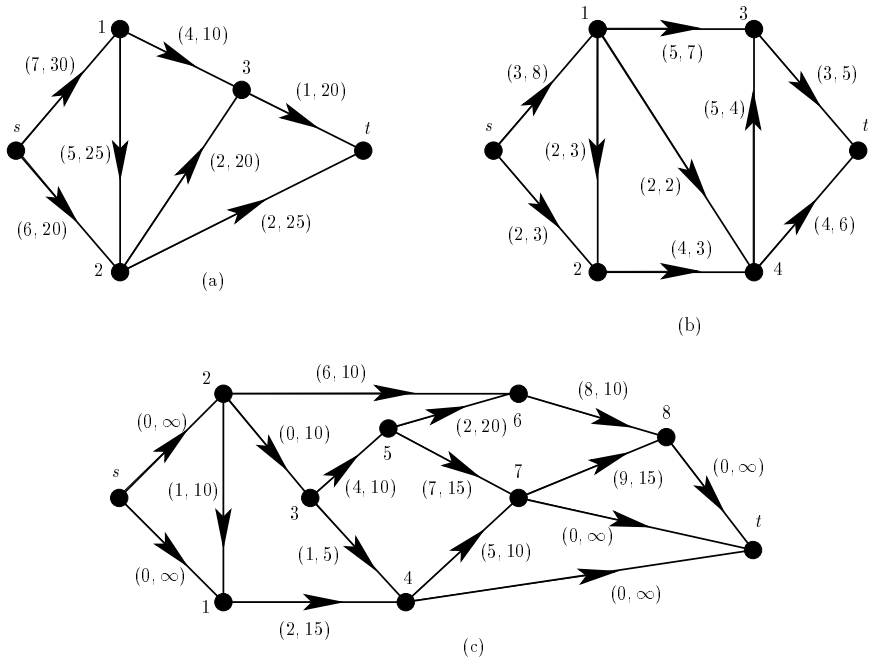


Figure 13.29: Minimum-cost flow problems for Exercise 2.

The Max-Flow Algorithm we have described in Section 13.3.5 is readily modified to give an efficient algorithm for solving the minimum-cost flow problem. (This algorithm will produce an integer solution if capacities, costs, and value v are integers.) For details, see, for example, Ahuja, Magnanti, and Orlin [1993], Lawler [1976], Minieka [1978], or Papadimitriou and Steiglitz [1982]. In special cases such as the transportation problem or the optimal assignment problem, there are more efficient algorithms.

EXERCISES FOR SECTION 13.4

1. In each directed network of Figure 13.20, consider the number in the circle as the cost rather than the flow, and consider the number in the square as the capacity as usual. In each case, find a minimum-cost flow of value 3 from s to t.
2. In each directed network of Figure 13.29, consider the ordered pair (a_{ij}, c_{ij}) on each arc (i, j) to be its cost (a_{ij}) and capacity (c_{ij}).
   (a) Find a minimum-cost flow of value 25 from s to t in directed network (a).
   (b) Find a minimum-cost flow of value 9 from s to t in directed network (b).
   (c) Find a minimum-cost flow of value 30 from s to t in directed network (c).

3. (a) Find a minimum-cost flow of value 10 for the building evacuation problem (Example 13.13) shown in Figure 13.28.

   (b) Explain your answer to part (a) in terms of how you would evacuate 10 people from the building described in Figure 13.28.

   (c) What is the largest number of people that can be evacuated from the building described in Figure 13.28? What is the minimum-cost flow for this value?

4. (Ahuja, Magnanti, and Orlin [1993]) An airline has 6 daily flights from New York to Chicago. They leave every two hours starting at 7 AM. The planes used on the morning flights have a 100-person capacity. After noon, the planes used have a 150-person capacity. The airline can "bump" overbooked passengers onto a later flight. After 5 PM, the airline can put a passenger on another airline's 10 PM flight, on which there are always seats. Bumped passengers are compensated at the rate of \$200 plus \$20 per hour of delay on any delay over 2 hours. Suppose that on a certain day, the airline has bookings of 110, 160, 103, 149, 175, and 140 people on their 6 flights. Formulate the problem of minimizing compensation costs as a minimum-cost flow problem.

5. A caterer knows in advance that for the next n days, a_j napkins will be needed on the jth day, $j = 1, 2, \ldots, n$. For any given day, the caterer can buy new napkins or use laundered napkins. Laundering of napkins can be done either by quick service, which takes q days, or by slow service, which takes r days. The cost of a new napkin is c cents, the cost of laundering a napkin quickly is d cents, and the cost of laundering one slowly is e cents. Starting with no napkins, how would the caterer meet the napkin requirement with minimum cost? Set this up as a minimum-cost flow problem. (*Note:* An analogous problem, which predates this problem historically, involves aircraft maintenance, with either quick or slow overhaul of engines.)

6. (Ahuja, Magnanti, and Orlin [1993], Prager [1957]). An investor considers investments in gold in T future time periods. In each time period, he can buy, sell, or hold gold that he already owns. Suppose that in period i, he can buy at most α_i ounces of gold, hold at most β_i ounces of gold, and must sell at least γ_i ounces of gold (due to prior contracts), and assume that he must hold purchased gold at least until the next time period. The costs involved with investing in gold are per ounce purchase cost p_i and per ounce selling price s_i in period i and per ounce holding cost w_i during period i. How should the investor make purchases, sales, and holds during the T time periods in order to maximize profit? Formulate the problem as a minimum-cost flow problem.

7. In the investment problem of Exercise 6, suppose we assume that the various restrictions and costs are independent of i, but that if you buy gold, you must order it two time periods in advance, and if you decide to hold gold in any time period, you must hold it for three time periods. How would you analyze this investment problem? (*Hint*: Compare with Example 13.13, building evacuation.)

8. (Cook, *et al.* [1998]) Suppose that D is a weakly connected digraph. We can "duplicate" any arc $a = (u, v)$ of D at cost $c(a)$, where by duplicating we mean adding another arc from u to v. We can make as many duplicates of an arc as we like. We want to duplicate as few arcs as possible in order to obtain a multidigraph that has an eulerian closed path. Formulate this as a minimum-cost flow problem.

(A similar problem arises in connection with the "Chinese Postman" Problem of Section 12.7.1.)

9. Suppose that we have two warehouses and two markets. There are 10 spools of wire at the first warehouse and 14 at the second. Moreover, 13 are required at the first market and 11 at the second. If the following matrix gives the transportation costs, find the minimum-cost transportation schedule.

		Factory	
		1	2
Warehouse	1	100	84
	2	69	75

10. Consider the transportation problem (Example 13.11).

    (a) Show that if $\sum a_i < \sum b_j$, there is no shipping pattern that meets requirements (13.13) and (13.14).

    (b) If $\sum a_i > \sum b_j$, show that we may as well assume that $\sum a_i = \sum b_j$ by creating a new $(m+1)$st market, setting $b_{m+1} = \sum_{i=1}^{n} a_i - \sum_{j=1}^{m} b_j$, adding arcs from each warehouse to the new market, and letting the capacities of all new arcs be 0.

11. In a directed network with costs, modify the definition of an augmenting chain to allow it to start at a vertex other than s and end at a vertex other than t. Define the cost of an augmenting chain to be the sum of the costs of forward arcs minus the sum of the costs of backward arcs. An *augmenting circuit* is an augmenting chain that forms a circuit.

    (a) In each example for Exercise 1, find a flow of value 3 that *does not* have minimum cost and find a flow-augmenting circuit with negative cost.

    (b) Show that if a flow of value v is of minimum cost, it admits no flow-augmenting circuit with negative cost.

    (c) Show that if a flow of value v admits no flow-augmenting circuit of negative cost, the flow has minimum cost.

12. We can reduce a transportation problem (Example 13.11) to a maximum-weight matching problem (Section 12.1) as follows. Build a bipartite graph $G = (X, Y, E)$ of $2\sum a_i$ vertices, as follows. Let X consist of a_i copies of the ith warehouse, $i = 1, 2, \ldots, n$, and let Y consist of b_j copies of the jth market, $j = 1, 2, \ldots, m$. G has all possible edges between vertices in X and in Y. On the edge joining a copy of the ith warehouse to a copy of the jth market, place a weight equal to $K - a_{ij}$, where K is sufficiently large. Show that an optimal solution to the transportation problem corresponds to a maximum-weight matching in G.

REFERENCES FOR CHAPTER 13

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., "Network Flows," in G. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd (eds.), *Handbooks in Operations Research and Management Science*, Vol. 1: *Optimization*, North-Holland, Amsterdam, 1989, 211–369.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., "Some Recent Advances in Network Flows," *SIAM Review, 33* (1991), 175–219.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1993.

Aronson, J. E., "A Survey of Dynamic Network Flows," *Ann. Oper. Res., 20* (1989), 1–66.

Arratia, R., and Lander, E. S., "The Distribution of Clusters in Random Graphs," *Adv. Appl. Math., 11* (1990), 36–48.

Baritchi, A., Cook, D. J., and Holder, L. B., "Discovering Structural Patterns in Telecommunications Data," in *Proceedings of the Thirteenth Annual Florida AI Research Symposium*, 2000.

Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D., *Linear Programming and Network Flows*, 2nd ed., Wiley, New York, 1990.

Berge, C., and Ghouila-Houri, A., *Programming, Games and Transportation Networks*, Wiley, New York, 1965.

Bondy, J. A., and Murty, U. S. R., *Graph Theory with Applications*, American Elsevier, New York, 1976.

Borůvka, O., "O Jistém Problému Minimálním," *Práce Mor. Prírodoved. Spol. v Brne (Acta Soc. Sci. Nat. Moravicae), 3* (1926), 37–58. (a)

Borůvka, O., "Príspevěk k. Řešení Otázky Ekonomické Stavby Elektrovodních Sítí," *Elektrotech. Obzor, 15* (1926), 153–154. (b)

Chalmet, L. G., Francis, R. L., and Saunders, P. B., "Network Models for Building Evacuation," *Management Sci., 28* (1982), 86–105.

Cheriton, D., and Tarjan, R. E., "Finding Minimum Spanning Trees," *SIAM J. Comput., 5* (1976), 724–742.

Church, K., "Massive Data Sets and Graph Algorithms in Telecommunications Systems," Session on Mathematical, Statistical, and Algorithmic Problems of Very Large Data Sets, American Mathematical Society Meeting, San Diego, CA, January 1997.

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., and Schrijver, A., *Combinatorial Optimization*, Wiley, New York, 1998.

Czekanowski, J., "Zur Differentialdiagnose der Neandertalgruppe," *Korrespondenzbl. Dtsch. Ges. Anthrop., Ethn. Urg., 40* (1909), 44–47.

Czekanowski, J., "Objektive Kriterien in der Ethnologie," *Ebenda, 43* (1911), 71–75.

Czekanowski, J., "Das Typenfrequenzgesetz," *Anthropol. Anz., 5* (1928), 15–20.

Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs," *Numer. Math., 1* (1959), 269–271.

Edmonds, J., and Karp, R. M., "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *J. ACM, 19* (1972), 248–264.

Elias, P., Feinstein, A., and Shannon, C. E., "Note on Maximum Flow through a Network," *IRE Trans. Inf. Theory, IT-2* (1956), 117–119.

Ford, L. R., and Fulkerson, D. R., "Maximal Flow through a Network," *Canad. J. Math., 8* (1956), 399–404.

Ford, L. R., and Fulkerson, D. R., "A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem," *Canad. J. Math., 9* (1957), 210–218.

Ford, L. R., and Fulkerson, D. R., *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

Frank, C. R., "A Note on the Assortment Problem," *Management Science, 11* (1965),

724–726.

Frank, H., and Frisch, I. T., *Communication, Transportation and Flow Networks*, Addison-Wesley, Reading, MA, 1971.

Gabow, H. N., Galil, Z., Spencer, T., and Tarjan, R. E., "Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs," *Combinatorics, 6* (1986), 109–122.

Glover, F., and Klingman, D., "Network Applications in Industry and Government," *AIIE Trans., 9* (1977), 363–376.

Godehart, E., *Graphs as Structural Models*, 2nd ed., Friedr. Vieweg & Sohn, Braunschweig, Germany, 1990.

Goldman, A. J., "Discrete Mathematics in Government," lecture presented at SIAM Symposium on Applications of Discrete Mathematics, Troy, NY, June 1981.

Gondran, M., and Minoux, M., *Graphs and Algorithms*, Wiley, New York, 1984.

Gower, J. C., and Ross, G. J. S., "Minimum Spanning Trees and Single Linkage Cluster Cluster Analysis," *Applied Statistics, 18*, (1969), 54–64.

Graham, R. L., and Hell, P., "On the History of the Minimum Spanning Tree Problem," *Annals of the History of Computing, 7* (1985), 43–57.

Guénoche, A., Hansen, P., and Jaumard, B., "Efficient Algorithms for Divisive Hierarchical Clustering with the Diameter Criterion," *J. Classification, 8* (1991), 5–30.

Hansen, P., Frank, O., and Jaumard, B., "Maximum Sum of Splits Clustering," *J. Classification, 6* (1989), 177–193.

Hansen, P., and Jaumard, B., "Minimum Sum of Diameters Clustering," *J. Classification, 4* (1987), 215–226.

Hansen, P., Jaumard, B., and Mladenovic, N., "Minimum Sum of Squares Clustering in a Low Dimensional Space," *J. Classification, 15* (1998), 37–55.

Hu, T. C., *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969.

Iri, M., *Network Flows, Transportation and Scheduling*, Academic Press, New York, 1969.

Jarník, V., "O Jistém Problému Minimálním," *Práce Mor. Prírodoved Spol. v Brne (Acta Soc. Sci. Nat. Moravicae), 6* (1930), 57–63.

Johnson, D. B., "Priority Queues with Update and Minimum Spanning Trees," *Inf. Process. Lett., 4* (1975), 53–57.

Karger, D., "Information Retrieval: Challenges in Interactive-Time Manipulation of Massive Text Collections," Session on Mathematical, Statistical, and Algorithmic Problems of Very Large Data Sets, American Mathematical Society Meeting, San Diego, CA, January 1997.

Kershenbaum, A., and Van Slyke, R., Computing Minimum Spanning Trees Efficiently, ACM 72, *Proceedings of the Annual ACM Conference*, 1972, 518–527.

Kruskal, J. B., "On the Shortest Spanning Tree of a Graph and the Traveling Salesman Problem," *Proc. Amer. Math. Soc., 7* (1956), 48–50.

Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

Menger, K., "Zur allgemeinen Kurventheorie," *Fund. Math., 10* (1927), 96–115.

Minieka, E., *Optimization Algorithms for Networks and Graphs*, Dekker, New York, 1978.

Mirkin, B., *Mathematical Classification and Clustering*, Kluwer Academic Publishers, Dordrecht-Boston-London, 1996.

Neil, M. D., "Multivariate Assessment of Software Products," *Software Testing, Veri-*

fication & Reliability, 1 (1992), 17–37.

Nešetřil, J., Milková, E., and Nešetřilová, H., "Otakar Borůvka on Minimum Spanning Tree Problem: Translation of Both the 1926 Papers, Comments, History," *Discrete Math., 233* (2001), 3–36.

Papadimitriou, C. H., and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

Prager, W., "On Warehousing Problems," *Oper. Res., 5* (1957), 504–512.

Prim, R. C., "Shortest Connection Networks and Some Generalizations," *Bell Syst. Tech. J., 36* (1957), 1389–1401.

Ravindran, A., "On Compact Book Storage in Libraries," *Opsearch, 8* (1971), 245–252.

Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Trans. Software Engrg., 3* (1977), 85–93.

Tarjan, R. E., *Data Structures and Network Algorithms*, SIAM, Philadelphia, 1983.

Tarjan, R. E., "A Simple Version of Karzanov's Blocking Flow Algorithm," *Oper. Res. Lett., 2* (1984), 265–268. (a)

Tarjan, R. E., "Efficient Algorithms for Network Optimization," *Proceedings of the International Congress of Mathematicians*, (Warsaw, 1983) Vols. 1, 2 (1984), 1619–1635. (b)

White, L. S., "Shortest Route Models for the Allocation of Inspection Effort on a Production Line," *Management Sci., 15* (1969), 249–259.

Yao, A., "An $O(|E|\log\log|V|)$ Algorithm for Finding Minimum Spanning Trees," *Information Processing Letters, 4* (1975), 21–23.

Zahn, C. T., "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," *IEEE Trans. Computing, C-20* (1971), 68–86.