M.O. Ball et al., Eds., *Handbooks in OR & MS, Vol. 7*

Chapter 1

Applications of Network Optimization

Ravindra K. Ahuja
Department of Industrial and Management Engineering, I.I.T., Kanpur – 208 016, India

Thomas L. Magnanti, James B. Orlin
Sloan School of Management, M.I.T., Cambridge, MA 02139, U.S.A.

M.R. Reddy
Department of Industrial and Management Engineering, I.I.T., Kanpur – 208 016, India

1. Introduction

Highways, telephone lines, electric power systems, computer chips, water delivery systems, and rail lines: these physical networks, and many others, are familiar to all of us. In each of these problem settings, we often wish to send some good(s) (vehicles, messages, electricity, or water) from one point to another, typically as efficiently as possible – that is, along a shortest route or via some minimum cost flow pattern. Although these problems trace their roots to the work of Gustav Kirchhoff and other great scientists of the last century, the topic of network optimization as we know it today has its origins in the 1940's with the development of linear programming and, more broadly, optimization as an independent field of scientific inquiry, and with the parallel development of digital computers capable of performing massive computations. Since then, the field of network optimization has grown at an almost dizzying pace with literally thousands of scientific papers and multitudes of applications modeling a remarkably wide range of practical situations.

Network optimization has always been a core problem domain in operations research, as well as in computer science, applied mathematics, and many fields of engineering and management. The varied applications in these fields not only occur 'naturally' on some transparent physical network, but also in situations that apparently are quite unrelated to networks. Moreover, because network optimization problems arise in so many diverse problem contexts, applications are scattered throughout the literature in several fields. Consequently, it is sometimes difficult for the research and practitioner community to fully appreciate the richness and variety of network applications.

This chapter is intended to introduce many applications and, in doing so, to highlight the pervasiveness of network optimization in practice. Our coverage is

not intended to be encyclopedic, but rather attempts to demonstrate a range of applications, chosen because they are (i) 'core' models (e.g., a basic production planning model), (ii) depict a range of applications including such fields as medicine and the molecular biology that might not be familiar to many readers, and (iii) cover many basic model types of network optimization: (1) shortest paths; (2) maximum flows; (3) minimum cost flows; (4) assignment problems; (5) matchings; (6) minimum spanning trees; (7) convex cost flows; (8) generalized flows; (9) multicommodity flows; (10) the traveling salesman problem; and (11) network design. We present five applications for each of the core shortest paths, maximum flows, and minimum cost flow problems, four applications for each of the matching, minimum spanning tree, and traveling salesman problems, and three applications for each of the remaining problems.

The chapter describes the following 42 applications, drawn from the fields of operations research, computer science, the physical sciences, medicine, engineering, and applied mathematics:

1. System of difference constraints;
2. Telephone operator scheduling;
3. Production planning problems;
4. Approximating piecewise linear functions;
5. DNA sequence alignment;
6. Matrix rounding problem;
7. Baseball elimination problem;
8. Distributed computing on a two-processor computer;
9. Scheduling on uniform parallel machines;
10. Tanker scheduling;
11. Leveling mountainous terrain;
12. Reconstructing the left ventricle from X-ray projections;
13. Optimal loading of a hopping airplane;
14. Directed Chinese postman problem;
15. Racial balancing of schools;
16. Locating objects in space;
17. Matching moving objects;
18. Rewiring of typewriters;
19. Pairing stereo speakers;
20. Determining chemical bonds;
21. Dual completion of oil wells;
22. Parallel saving heuristics;
23. Measuring homogeneity of bimetallic objects;
24. Reducing data storage;
25. Cluster analysis;
26. System reliability bounds;
27. Urban traffic flows;
28. Matrix balancing;
29. Stick percolation problem;
30. Determining an optimal energy policy;

31. Machine loading;
32. Managing warehousing goods and funds flow;
33. Routing of multiple commodities;
34. Racial balancing of schools;
35. Multivehicle tanker scheduling;
36. Manufacturing of printed circuit boards;
37. Identifying time periods for archeological finds;
38. Assembling physical mapping in genetics;
39. Optimal vane placement in turbine engines;
40. Designing fixed cost communication and transportation systems;
41. Local access telephone network capacity expansion;
42. Multi-item production planning.

In addition to these 42 applications, we provide references for 140 additional applications.

2. Preliminaries

In this section, we introduce some basic notation and definitions from graph theory as well as a mathematical programming formulation of the minimum cost flow problem, which is the core network flow problem that lies at the heart of network optimization. We also present some fundamental transformations that we frequently use while modeling applications as network problems.

Let $G = (N, A)$ be a directed network defined by a set N of n *nodes,* and a set A of m *directed arcs*. Each arc $(i, j) \in A$ has an associated *cost* c_{ij} per unit flow on that arc. We assume that the flow cost varies linearly with the amount of flow. Each arc $(i, j) \in A$ also has a *capacity* u_{ij} denoting the maximum amount that can flow on the arc, and a *lower bound* l_{ij} denoting the minimum amount that must flow on the arc. We associate with each node $i \in N$ an integer $b(i)$ representing its supply/demand. If $b(i) > 0$, then node i is a *supply node*; if $b(i) < 0$, then node i is a *demand node*; and if $b(i) = 0$, then node i is a *transshipment node.*

The minimum cost flow problem is easy to state: we wish to determine a least cost shipment of a commodity through a network that will satisfy the flow demands at certain nodes from available supplies at other nodes. The decision variables in the minimum cost flow problem are arc flows; we represent the flow on an arc $(i, j) \in A$ by x_{ij}. The minimum cost flow problem is an optimization model formulated as follows:

$$\text{minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i), \qquad \text{for all } i \in N, \tag{1b}$$

$$l_{ij} \le x_{ij} \le u_{ij}, \qquad \text{for all } (i, j) \in A. \tag{1c}$$

The data for this model satisfies the feasibility condition $\sum_{i=1}^{n} b(i) = 0$ (that is, total supply must equal the total demand). We refer to the constraints in (1b) as *mass balance constraints*. The mass balance constraints state that the net flow out of each node (outflow minus inflow) must equal the supply/demand of the node. The flow must also satisfy the lower bound and capacity constraints (1c) which we refer to as the *flow bound constraints*. The flow bounds typically model physical capacities or restrictions imposed upon the flows' operating ranges. In most applications, the lower bounds on arc flows are zero; therefore, if we do not state lower bounds explicitly, we assume that they have value zero.

We now collect together several basic definitions and describe some notation. A *walk* in $G = (N, A)$ is a sequence of nodes and arcs i_1, (i_1, i_2), i_2, (i_2, i_3), i_3, $\ldots$, (i_{r-1}, i_r), i_r satisfying the property that either $(i_k, i_{k+1}) \in A$ or $(i_{k+1}, i_k) \in A$ for each $k = 1, \ldots, r-1$. A walk might revisit nodes. A *path* is a walk whose nodes (and, hence, arcs) are all distinct. For simplicity, we often refer to a path as a sequence of nodes i_1–i_2–$\ldots$–i_k when its arcs are apparent from the problem context. A *directed path* is defined similarly except that for any two consecutive nodes i_k and i_{k+1} on the path, the path must contain the arc (i_k, i_{k+1}). A *directed cycle* is a directed path together with the arc (i_r, i_1), and a *cycle* is a path together with the arc (i_r, i_1) or (i_1, i_r).

A graph $G' = (N', A')$ is a *subgraph* of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$. A graph $G' = (N', A')$ is a spanning subgraph of $G = (N, A)$ if $N' = N$ and $A' \subseteq A$. Two nodes i and j are said to be *connected* if the graph contains at least one undirected path between these nodes. A graph is said to be *connected* if every pair of its nodes are connected; otherwise, it is *disconnected*. The connected subgraphs of a graph are called its *components*. A tree is a connected graph that contains no cycle. A subgraph T is a *spanning tree* of G if T is a tree of G containing all its nodes. A *cut* of G is any set $Q \subseteq A$ satisfying the property that the graph $G' = (N, A - Q)$ is disconnected, and no subset of Q has this property. A cut partitions the graph into two sets of nodes, X and N–X. We shall sometimes represent the cut Q as the node partition $[X, N\text{–}X]$. A cut $[X, N\text{–}X]$ is an s–t cut for two specially designated nodes s and t if $s \in X$ and $t \in N$–X.

Transformations

Frequently, we require network transformations to model an application context, to simplify a network problem, or to show equivalencies between different network problems. We now briefly describe some of these transformations.

Removing undirected networks. Sometimes minimum cost flow problems contain undirected arcs. Whereas a directed arc (i, j) permits flow only from node i to node j, an undirected arc $\{i, j\}$ permits flow from node i to node j as well as flow from node j to node i. To transform the undirected case into the directed case, we replace each undirected arc $\{i, j\}$ of cost c_{ij} and capacity u_{ij}, by two directed arcs (i, j) and (j, i), both of cost c_{ij} and capacity u_{ij}. It is easy to see that if x_{ij} and x_{ji} are the flows on arcs (i, j) and (j, i) in the directed network respectively,

then $x_{ij} - x_{ji}$ or $x_{ji} - x_{ij}$, whichever is nonnegative, is the associated flow on the undirected arc $\{i, j\}$.

Removing nonzero lower bounds. Suppose arc (i, j) has a nonzero lower bound l_{ij} on its flow x_{ij}. We can eliminate this lower bound by sending l_{ij} units of flow on arc (i, j), which decreases $b(i)$ by l_{ij} units and increases $b(j)$ by l_{ij} units, and then we measure (by the variable x'_{ij}) the incremental flow on the arc beyond the flow value l_{ij} (therefore, we reduce the capacity of the arc (i, j) to $u_{ij} - l_{ij}$).

Node splitting. Node splitting transforms each node i into two nodes i' and i'' corresponding to the node's output and input functions. This transformation replaces each original arc (i, j) by an arc (i', j'') of the same cost and capacity. It also adds an arc (i'', i') of zero cost and with infinite capacity for each i. The input side of node i (i.e., node i'') receives all the node's inflow, the output side (i.e., node i') sends all the node's outflow, and the additional arc (i'', i') carries flow from the input side to the output side. We define the supplies/demands of nodes in the transformed network in accordance with the following two cases: (i) if $b(i) \geq 0$, then $b(i'') = b(i)$ and $b(i') = 0$; and (ii) if $b(i) < 0$, then $b(i'') = 0$ and $b(i') = b(i)$. It is easy to show a one-to-one correspondence between a flow in the original network and the corresponding flow in the transformed network.

We can use the node splitting transformation to handle situations in which nodes as well as arcs have associated capacities and costs. In these situations, each unit of flow passing through node i incurs a cost c_i and the maximum flow that can pass through the node is u_i. We can reduce this problem to the standard 'arc flow' form of the network flow problem by performing the node splitting transformation and letting c_i and u_i be the cost and capacity of arc (i'', i').

3. Shortest paths

The shortest path problem is among the simplest network flow problems. For this problem, we wish to find a path of minimum cost (length) from a specified *source node* s to another specified *sink node* t in either a directed or undirected network assuming that each arc $(i, j) \in A$ has an associated cost (or length) c_{ij}. In the formulation of the minimum cost flow problem given in (1) for a directed network, if we set $b(s) = 1$, $b(t) = -1$, and $b(i) = 0$ for all other nodes, and set each $l_{ij} = 0$ and each $u_{ij} \geq 1$, then the solution to this problem will send one unit of flow from node s to node t along the shortest directed path. If we want to determine shortest paths from the source node s to every other node in the network, then in the minimum cost flow problem, we set $b(s) = (n - 1)$ and $b(i) = -1$ for all other nodes. We can set each arc capacity u_{ij} to $n - 1$. The minimum cost flow solution would then send one unit of flow from node s to every other node i along a shortest path. We will consider several applications defined on directed networks. We can model problems defined on undirected

networks as special cases of the minimum cost network flow problem by using the transformations described in Section 2.

Shortest path problems are alluring to both researchers and to practitioners for several reasons: (i) they arise frequently in practice since in a wide variety of application settings we wish to send some material (for example, a computer data packet, a telephone call, or a vehicle) between two specified points in a network as quickly, as cheaply, or as reliably as possible; (ii) they are easy to solve efficiently; (iii) as the simplest network models, they capture many of the most salient core ingredients of network flows and so they provide both a benchmark and a point of departure for studying more complex network models; and (iv) they arise frequently as subproblems when solving many combinatorial and network optimization problems. In this section, we describe a few applications of the shortest path problem that are indicative of its range of applications. The applications arise in applied mathematics, biology, computer science, production planning, and work force scheduling. We conclude the section with a set of references for many additional applications in a wide variety of fields.

Application 1. System of difference constraints [Bellman, 1958]

In some linear programming applications (see Application 2) with constraints of the form $\mathbf{A}x \leq b$, the $n \times m$ constraint matrix $\mathbf{A}$ contains one $+1$ and one -1 in each row; all the other entries are zero. Suppose that the kth row has a $+1$ entry in column j_k and a -1 entry in column i_k; the entries in the vector b have arbitrary signs. This linear program defines the following set of m *difference constraints* in the n variables $x = (x(1), x(2), \ldots, x(n))$:

$$x(j_k) - x(i_k) \leq b(k), \quad \text{for each } k = 1, \ldots, m. \tag{2}$$

We wish to determine whether the system of difference constraints given by (2) has a feasible solution, and if so, we want to identify one. This model arises in a variety of applications; Application 2 describes the use of this model in the telephone operator scheduling; additional applications arise in the scaling of data [Orlin & Rothblum, 1985] and just-in-time scheduling [Elmaghraby, 1978; Levner & Nemirovsky, 1991].

Each system of difference constraints has an associated graph G, which we call a *constraint graph*. The constraint graph has n nodes corresponding to the n variables and m arcs corresponding to the m difference constraints. We associate an arc (i_k, j_k) of length $b(k)$ in G with the constraint $x(j_k) - x(i_k) \leq b(k)$. As an example, consider the following system of constraints:

$$x(3) - x(4) \leq 5, \tag{3a}$$

$$x(4) - x(1) \leq -10, \tag{3b}$$

$$x(1) - x(3) \leq 8, \tag{3c}$$

$$x(2) - x(1) \leq -11, \tag{3d}$$

$$x(3) - x(2) \leq 2. \tag{3e}$$

To model the system of difference constraints as a shortest path problem, we use the two following well-known results about shortest paths (see, e.g., Cormen, Leiserson & Rivest [1990], and Ahuja, Magnanti & Orlin [1993]):

Observation 1. The shortest path distance $d(i)$ from source node s to node i for every node $i \in N$ satisfy the following optimality conditions: $d(j) - d(i) \leq c_{ij}$ for every arc $(i, j) \in A$.

Observation 2. The shortest path distances in a network G exist if and only if G contains no negative cycle (i.e., a cycle whose total cost, summed over all its arcs, is negative).

First, notice that the structure of the shortest path optimality conditions given in Observation 1 is similar to the inequalities of the system of difference constraints (3). In fact, Figure 1a gives the network for which (3) become the shortest path optimality conditions. The second observation implies that the system of difference constraints has a feasible solution if and only if the corresponding network contains no negative cycle. For instance, the network shown in Figure 1a contains a negative cycle 1–2–3–1 of length −1, and the corresponding constraints (i.e., $x(2) - x(1) \leq -11$, $x(3) - x(2) \leq 2$, and $x(1) - x(3) \leq 8$) are inconsistent because summing these constraints yields $0 \leq -1$. We can thus conclude that the system of difference constraints given by (3) has no feasible solution.

We can detect the presence of a negative cycle in a network by using a label correcting algorithm. Label correcting algorithms require that all the nodes in the network are reachable by a directed path from some node, which we use as the source node for the shortest path problem. To satisfy this requirement, we introduce a new node s and join it to all the nodes in the network with arcs of zero cost. For our example, Figure 1b shows the modified network. Since all the arcs incident to node s are directed out of this node, node s is not contained in any

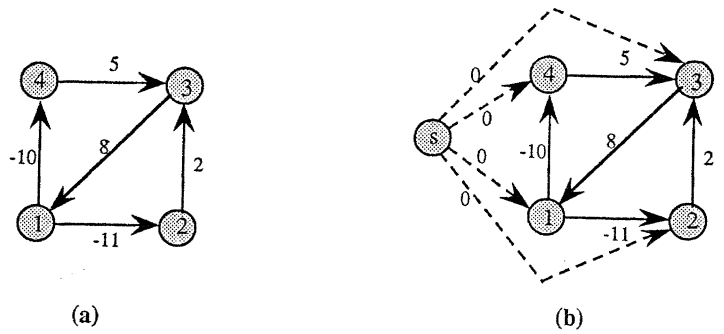


Fig. 1. Graph corresponding to a system of difference constraints.

directed cycle and so the modification does not create any new directed cycles, and so does not introduce any cycles with negative costs. Label correcting algorithms either detect the presence of a negative cycle or provide the shortest path distances. In the former case, the system of difference constraints has no solution, and in the latter case, the shortest path distances constitute a solution of (2).

Application 2. Telephone operator scheduling [Bartholdi, Orlin & Ratliff, 1980]

The following telephone operator scheduling problem is an application of the system of difference constraints. A telephone company needs to schedule operators around the clock. Let $b(i)$ for $i = 0, 1, 2, \ldots, 23$, denote the minimum number of operators needed for the ith hour of the day ($b(0)$ denotes number of operators required between midnight and 1 AM). Each telephone operator works in a shift of 8 consecutive hours and a shift can begin at any hour of the day. The telephone company wants to determine a 'cyclic schedule' that repeats daily, i.e., the number of operators assigned to the shift starting at 6 AM and ending at 2 PM is the same for each day. The optimization problem requires that we identify the fewest operators needed to satisfy the minimum operator requirement for each hour of the day. If we let y_i denote the number of workers whose shift begins at the ith hour, then we can state the telephone operator scheduling problem as the following optimization model:

$$\text{minimize} \quad \sum_{i=0}^{23} y_i \tag{4a}$$

subject to

$$y_{i-7} + y_{i-6} + \ldots + y_i \geq b(i), \qquad \text{for all } i = 8 \text{ to } 23, \tag{4b}$$

$$y_{17+i} + \ldots + y_{23} + y_0 + \ldots + y_i \geq b(i), \qquad \text{for all } i = 0 \text{ to } 7, \tag{4c}$$

$$y_i \geq 0 \qquad \text{for all } i = 0 \text{ to } 23. \tag{4d}$$

Notice that this linear program has a very special structure because the associated constraint matrix contains only 0's and 1's and the 1's or 0's in each row appear consecutively. In this application, we study a restricted version of the telephone operator scheduling problem: we wish to determine whether some feasible schedule uses p or fewer operators. We convert this restricted problem into a system of difference constraints by redefining the variables. Let $x(0) = y_0$, $x(1) = y_0 + y_1$, $x(2) = y_0 + y_1 + y_2, \ldots$, and $x(23) = y_0 + y_2 + \ldots + y_{23} = p$. In terms of these new variables, we can rewrite each constraint in (4b) as

$$x(i) - x(i-8) \geq b(i), \qquad \text{for all } i = 8 \text{ to } 23, \tag{5a}$$

and each constraints in (4c) as

$$x(23) - x(16+i) + x(i) = p - x(16-i) + x(i) \geq b(i), \qquad \text{for all } i = 0 \text{ to } 7. \tag{5b}$$

Finally, the nonnegativity constraints (4d) become

$$x(i) - x(i-1) \geq 0. \tag{5c}$$

By virtue of this transformation, we have reduced the restricted version of the telephone operator scheduling problem into a problem of finding a feasible solution of the system of difference constraints. Application 1 shows that we can further reduce this problem into a shortest path problem.

Application 3. Production planning problems [Veinott & Wagner, 1962; Zangwill, 1969; Evans, 1977]

Many optimization problems in production and inventory planning are network optimization models. All of these models address a basic *economic order quantity* issue: when we plan a production run of any particular product, how much should we produce? Producing in large quantities reduces the time and cost required to set up equipment for the individual production runs; on the other hand, producing in large quantities also means that we will carry many items in inventory awaiting purchase by customers. The *economic order quantity* strikes a balance between the set up and inventory costs to find the production plan that achieves the lowest overall costs. The models that we consider in this section all attempt to balance the production and inventory carrying costs while meeting known demands that vary throughout a given planning horizon. We present one of the simplest models: a single product, single stage model with concave costs and backordering, and transform it to a shortest path problem. This is an example not naturally as a shortest path problem, but that becomes a shortest path problem because of an underlying 'spanning tree property' of the optimal solution.

In this model, we assume that the production cost in each period is a concave function of the level of production. In practice, the production x_j in the jth period frequently incurs a fixed cost F_j (independent of the level of production) and a per unit production cost c_j. Therefore, for each period j, the production cost is 0 for $x_j = 0$, and $F_j + c_j x_j$ if $x_j > 0$, which is a concave function of the production level x_j. The production cost might also be concave due to other economies of scale in production. In this model, we also permit backordering, which implies that we might not fully satisfy the demand of any period from the production in that period or from current inventory, but could fulfill the demand from production in future periods. We assume that we do not lose any customer whose demand is not satisfied on time and who must wait until his or her order materializes. Instead, we incur a penalty cost for backordering any item. We assume that the inventory carrying and backordering costs are linear, and that we have no capacity imposed upon production, inventory, or backordering volumes.

In this model, we wish to meet a prescribed demand d_j for each of K periods $j = 1, 2, \ldots, K$ by either producing an amount x_j in period j, by drawing upon the inventory I_{j-1} carried from the previous period, and/or by backordering the item from the next period. Figure 2a shows the network for modeling this problem. The network has $K + 1$ nodes: the jth node, for $j = 1, 2, \ldots, K$, represents the

jth planning period; node 0 represents the 'source' of all production. The flow on the *production arc* $(0, j)$ prescribes the production level x_j in period j, the flow on *inventory carrying arc* $(j, j+1)$ prescribes the inventory level I_j to be carried from period j to period $j+1$, and the flow B_j on the backordering arc $(j, j-1)$ represents the amount backordered from the next period.

The network flow problem in Figure 2a is a concave cost flow problem, because the cost of flow on every production arc is a concave function. The following well-known result about concave cost flow problems helps us to solve the problem (see, for example, Ahuja, Magnanti & Orlin [1993]):

Spanning tree property. A concave cost network flow minimization problem whose objective function is bounded from below over the set of feasible solutions always has an *optimal spanning tree solution*, that is, an optimal solution in which only the arcs in a spanning tree have positive flow (all other arcs, possibly including some arcs in the spanning tree, have zero flow).

Figure 2b shows an example of a spanning tree solution. This result implies the following property, known as the *production property*: In the optimal solution, each time we produce, we produce enough to meet the demand for an integral number of contiguous periods. Moreover, in no period do we both produce and carry inventory from the previous period or into next period.

The production property permits us to solve the production planning problem very efficiently as a shortest path problem on an auxiliary network G' shown in Figure 2c, which is defined as follows: The network G' consists of nodes 1 through $K+1$ and contains an arc (i, j) for every pair of nodes i and j with $i < j$. We set the cost of arc (i, j) equal to the sum of the production, inventory carrying and backorder carrying costs incurred in satisfying the demands of periods $i, i+1, \ldots, j-1$ by producing in some period k between i and $j-1$; we select this period k that gives the least possible cost. In other words, we vary k from i to $j-1$, and for each k, we compute the cost incurred in satisfying the demands of periods i through $j-1$ by the production in period k; the minimum of these values defines the cost of arc (i, j) in the auxiliary network G'. Observe that for every production schedule satisfying the production property, G' contains a directed path from node 1 to node $K+1$ with the same objective function value, and vice-versa. Therefore, we can obtain the optimal production schedule by solving a shortest path problem.

Several variants of the production planning problem arise in practice. If we impose capacities on the production, inventory, or backordering arcs, then the production property does not hold and we cannot formulate this problem as a shortest path problem. In this case, however, if the production cost in each period is linear, the problem becomes a minimum cost flow model. The minimum cost flow problem also models multistage situations in which a product passes through a sequence of operations. To model this situation, we would treat each production operation as a separate stage and require that the product pass through each of the stages before completing its production. In a further multi-item generalization, common manufacturing facilities are used to manufacture multiple products in

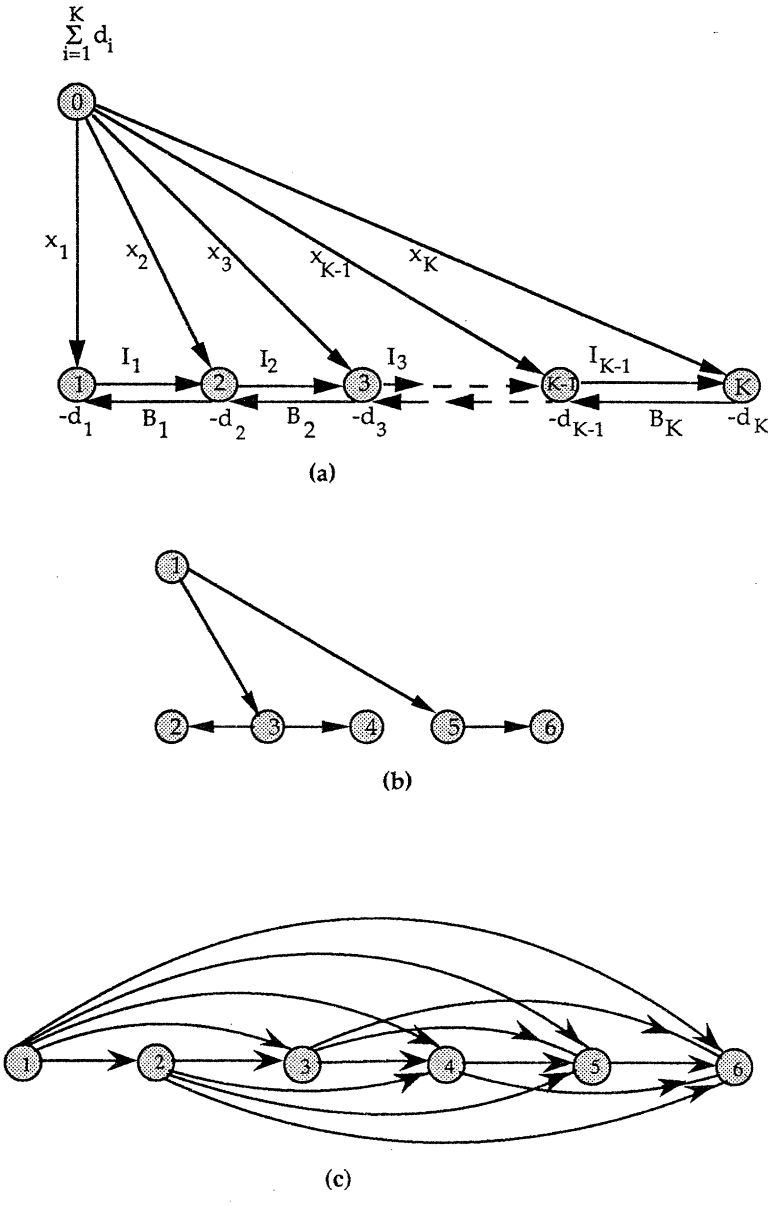


Fig. 2. Production planning problem. (a) Underlying network. (b) Graphical structure of a spanning tree solution. (c) The resulting shortest path problem.

multiple stages; this problem is a multicommodity flow problem. The references cited for this application describe these various generalizations.

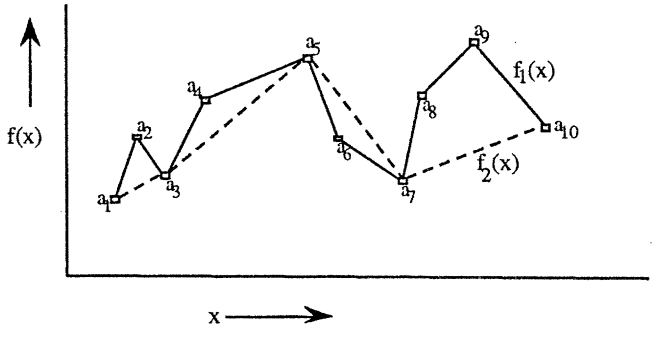
Application 4. Approximating piecewise linear functions [Imai & Iri, 1986]

Numerous applications encountered within many different scientific fields use piecewise linear functions. On several occasions, because these functions contain a large number of breakpoints, they are expensive to store and to manipulate (for

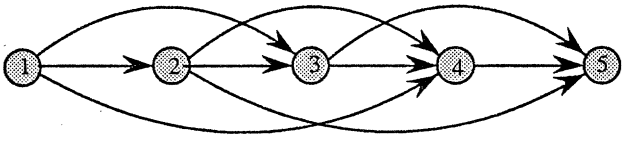
example, even to evaluate). In these situations, it might be advantageous to replace the piecewise linear function by another approximating function that uses fewer breakpoints. By approximating the function, we will generally be able to save on storage space and on the cost of using the function; we will, however, incur a cost because the approximating function introduces inaccuracies in representing the function. In making the approximation, we would like to make the best possible tradeoff between these conflicting costs and benefits.

Let $f_1(x)$ be a piecewise linear function of a scalar x. We represent the function in the two-dimensional plane: it passes through n points $a_1 = (x_1, y_1)$, $a_2 = (x_2, y_2), \ldots, a_n = (x_n, y_n)$. Suppose that we have ordered the points so that $x_1 \leq x_2 \leq \cdots \leq x_n$. We assume that the function varies linearly between every two consecutive points x_i and x_{i+1}. We consider situations in which n is very large and for practical reasons we wish to approximate the function $f_1(x)$ by another function $f_2(x)$ that passes through only a subset of the points $a_1, a_2, \ldots, a_n$ (but including a_1 and a_n). As an example, consider Figure 3a: in this figure, we have approximated a function $f_1(x)$ passing through 10 points by a function $f_2(x)$ (drawn with dotted lines) passing through only 5 of the points.

This approximation results in a savings in storage space and in the use (evaluation) of the function. Suppose that the cost of storing one data point is a. Therefore, by storing fewer data points, we incur less storage costs. But the



(a)



(b)

Fig. 3. Approximating precise linear functions. (a) Approximating a function $f_1(x)$ passing through 10 points by a function $f_2(x)$ passing through only 5 points. (b) Corresponding shortest path problem.

approximation introduces errors with an associated penalty. We assume that the error of an approximation is proportional to the sum of the squared errors between the actual data points and the estimated points. In other words, if we approximate the function $f_1(x)$ by $f_2(x)$, then the penalty is

$$\beta \sum_{k=1}^{p} [f_1(x_k) - f_2(x_k)]^2 \tag{6}$$

for some constant β. Our decision problem is to identify the subset of points to be used to define the approximation function $f_2(x)$ so we incur the minimum total cost as measured by the sum of the cost of storing and the cost of the errors incurred by the approximation.

We formulate this problem as a shortest path problem on a network G with n nodes, numbered 1 through n, as follows. The network contains an arc (i, j) for each pair of nodes i and j. Figure 3b gives an example of the network with $n = 5$ nodes. The arc (i, j) in this network signifies that we approximate the linear segments of the function $f_1(x)$ between the points $a_i, a_{i+1}, \ldots, a_j$ by one linear segment joining the points a_i and a_j. Each directed path from node 1 to node n in G corresponds to a function $f_2(x)$, and the cost of this path equals the total cost for storing this function and for using it to approximate the original function. For example, the path 1–3–5 corresponds to the function $f_2(x)$ passing through the points a_1, a_3 and a_5. The cost c_{ij} of the arc (i, j) has two components: the storage cost a and the penalty associated with approximating all the points between a_i and a_j by the line joining a_i and a_j. Observe that the coordinates of a_i and a_j are given by $[x_i, y_i]$ and $[x_j, y_j]$. The function $f_2(x)$ in the interval $[x_i, x_j]$ is given by the line $f_2(x) = (x - x_i)\{[f_1(x_j) - f_1(x_i)]/(x_j - x_i)\}$. This interval contains the data points with x-coordinates as $x_i, x_{i+1}, \ldots, x_j$, and so we must associate the corresponding terms of (6) with the cost of the arc (i, j). Consequently, we define the cost c_{ij} of an arc (i, j) as

$$c_{ij} = \alpha + \beta \sum_{k=i}^{j} [f_1(x_k) - f_2(x_k)]^2.$$

As a consequence of the preceding observations, we see that the shortest path from node 1 to node n specifies the optimal set of points needed to define the approximating function $f_2(x)$.

Application 5. DNA sequence alignment [Waterman, 1988]

Scientists model strands of DNA as a sequence of letters drawn from the alphabet {A,C,G,T}. Given two sequences of letters, say $B = b_1 b_2 \ldots b_p$ and $D = d_1 d_2 \ldots d_q$ of possibly different lengths, molecular biologists are interested in determining how similar or dissimilar these sequences are to each other. (These sequences are subsequences of the nucleic acids of DNA in a chromosome typically containing several thousand letters.) A natural way of measuring the dissimilarity between the two sequences B and D is to determine the minimum

$$\begin{array}{lllllll} B' = & \oplus & \oplus & A & G & T & \\ D = & C & T & A & G & C & \end{array} \Longrightarrow \begin{array}{lllll} C & T & A & G & C \\ C & T & A & G & C \end{array}$$

Fig. 4. Transforming the sequence B into the sequence D.

'cost' required to transform sequence B into sequence D. To transform B into D, we can perform the following operations: (i) insert an element into B (at any place in the sequence) at a 'cost' of a units; (ii) delete an element from B (at any place in the sequence) at a 'cost' of b units; and (iii) mutate an element b_i into an element d_j at a 'cost' of $g(b_i, d_j)$ units. Needless to say, it is possible to transform the sequence B into the sequence D in many ways and so identifying a minimum cost transformation is a nontrivial task. We show how we can solve this problem using dynamic programming, which we can also view as solving a shortest path problem on an appropriately defined network.

Suppose that we conceive of the process of transforming the sequence B into the sequence D as follows. First, add or delete elements from the sequence B so that the modified sequence, say B', has the same number of elements as D. Next 'align' the sequences B' and D to create a one-to-one alignment between their elements. Finally, mutate the elements in the sequence B' so that this sequence becomes identical with the sequence D. As an example, suppose that we wish to transform the sequence B = AGTT into the sequence D = CTAGC. One possible transformation is to delete one of the elements T from B and add two new elements at the beginning, giving the sequence $B' = \oplus \oplus$ AGT (we denote any new element by a placeholder $\oplus$ and later assign a letter to this placeholder). We then align B' with D, as shown in Figure 4, and mutate the element T into C so that the sequences become identical. Notice that because we are free to assign values to the newly added elements, they do not incur any mutation cost. The cost of this transformation is $b + 2a + g$(T,C).

We now describe a dynamic programming formulation of this problem. Let $f(i, j)$ denote the minimum cost of transforming the subsequence $b_1 b_2 \ldots b_i$ into the subsequence $d_1 d_2 \ldots d_j$. We are interested in the value $f(p, q)$, which is the minimum cost of transforming B into D. To determine $f(p, q)$, we will determine $f(i, j)$ for all $i = 0, 1, \ldots, p$, and for all $j = 0, 1, \ldots, q$. We can determine these intermediate quantities $f(i, j)$ using the following recursive relationships:

$$f(i, 0) = \beta i \quad \text{for all } i; \tag{7a}$$

$$f(0, j) = \alpha j \quad \text{for all } j; \text{ and} \tag{7b}$$

$$f(i, j) = \min\{f(i-1, j-1) + g(b_i, d_j),\ f(i, j-1) + \alpha,\ f(i-1, j) + \beta\}. \tag{7c}$$

We now justify this recursion. The cost of transforming a sequence of i elements into a null sequence is the cost of deleting i elements. The cost of transforming a null sequence into a sequence of j elements is the cost of adding j elements.

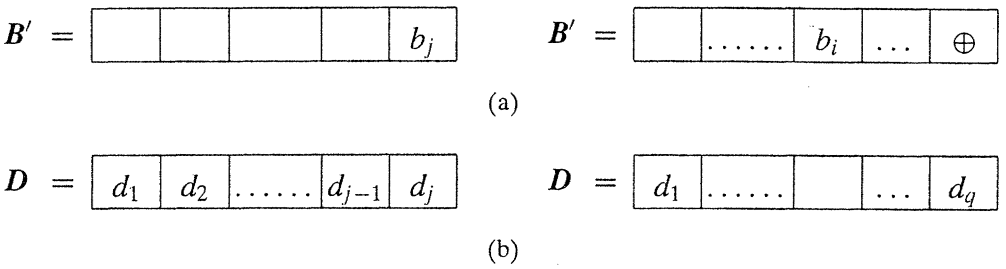


Fig. 5. Explaining the dynamic programming recursion.

Next consider $f(i, j)$. Let B' denote the optimal aligned sequence of B (i.e., the sequence just before we create the mutation of B' to transform it into D). At this point, B' satisfies exactly one of the following three cases:

Case 1. *B' contains the letter b_i which is aligned with the letter d_j of D* (as shown in Figure 5a). In this case, $f(i, j)$ equals the optimal cost of transforming the subsequence $b_1 b_2 \ldots b_{i-1}$ into $d_1 d_2 \ldots d_{j-1}$ and the cost of transforming the element b_i into d_j. Therefore, $f(i, j) = f(i-1, j-1) + g(b_i, d_j)$.

Case 2. *B' contains the letter b_i which is not aligned with the letter d_j* (as shown in Figure 5b). In this case, b_i is to the left of d_j and so a newly added element must be aligned with b_j. As a result, $f(i, j)$ equals the optimal cost of transforming the subsequence $b_1 b_2 \ldots b_i$ into $d_1 d_2 \ldots d_{j-1}$ plus the cost of adding a new element to B. Therefore, $f(i, j) = f(i, j-1) + \alpha$.

Case 3. *B' does not contains the letter b_i*. In this case, we must have deleted b_i from B and so the optimal cost of the transformation equals the cost of deleting this element and transforming the remaining sequence into D. Therefore, $f(i, j) = f(i-1, j) + \beta$.

The preceding discussion justifies the recursive relationships specified in (7). We can use these relationships to compute $f(i, j)$ for increasing values of i and, for a fixed value of i, for increasing values of j. This method allows us to compute $f(p, q)$ in $O(pq)$ time, that is, time proportional to the product of the number of elements in the two sequences.

We can alternatively formulate the DNA sequence alignment problem as a shortest path problem. Figure 6 shows the shortest path network for this formulation for a situation with $p = 3$ and $q = 3$. For simplicity, in this network we let g_{ij} denote $g(b_i, d_j)$. We can establish the correctness of this formulation by applying an induction argument based upon the induction hypothesis that the shortest path length from node 0^0 to node i^j equals $f(i, j)$. The shortest path from node 0^0 to node i^j must contain one of the following arcs as the last arc in the path: (i) arc $(i-1^{j-1}, i^j)$; (ii) arc (i^{j-1}, i^j), or (iii) arc $(i-1^j, i^j)$. In these three cases, the lengths of these paths will be $f(i-1, j-1) + g(b_i, d_j)$, $f(i, j-1) + \alpha$, and $f(i-1, j) + \beta$. Clearly, the shortest path length $f(i, j)$ will

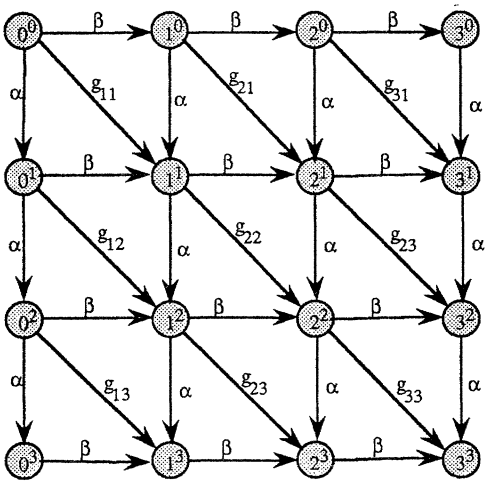


Fig. 6. Sequence alignment problem as a shortest path problem.

equal the minimum of these three numbers, which is consistent with the dynamic programming relationships stated in (4).

This application shows a relationship between shortest paths and dynamic programming. We have seen how to solve the DNA sequence alignment problem through the dynamic programming recursion or by formulating and solving it as a shortest path problem on an acyclic network. The recursion we use to solve the dynamic programming problem is just a special implementation of one of the standard algorithms for solving shortest path problems on acyclic networks. This observation provides us with a concrete illustration of the meta-statement that '(deterministic) dynamic programming is a special case of the shortest path problem'. Accordingly, shortest path problems model the enormous range of applications in many disciplines that are solvable by dynamic programming.

Additional applications

Some additional applications of the shortest path problem include: (1) *knapsack problem* [Fulkerson, 1966]; (2) *tramp steamer problem* [Lawler, 1966]; (3) *allocating inspection effort on a production line* [White, 1969]; (4) *reallocation of housing* [Wright, 1975]; (5) *assortment of steel beams* [Frank, 1965]; (6) *compact book storage in libraries* [Ravindran, 1971]; (7) *concentrator location on a line* [Balakrishnan, Magnanti, Shulman & Wong, 1991]; (8) *manpower planning problem* [Clark & Hasting, 1977]; (9) *equipment replacement* [Veinott & Wagner, 1962]; (10) *determining minimum project duration* [Elmaghraby, 1978]; (11) *assembly line balancing* [Gutjahr & Nemhauser, 1964]; (12) *optimal improvement of transportation networks* [Goldman & Nemhauser, 1967]; (13) *machining process optimization* [Szadkowski, 1970]; (14) *capacity expansion* [Luss, 1979]; (15) *routing in computer communication networks* [Schwartz & Stern, 1980]; (16) *scaling of matrices*

[Golitschek & Schneider, 1984]; (17) *city traffic congestion* [Zawack & Thompson, 1987]; (18) *molecular confirmation* [Dress & Havel, 1988]; (19) *order picking in an isle* [Goetschalckx & Ratliff, 1988]; and (20) *robot design* [Haymond, Thornton & Warner, 1988]. Shortest path problems often arise as important subroutines within algorithms for solving many different types of network optimization problems. These applications are too numerous to mention.

4. Maximum flows

In the maximum flow problem we wish to send the maximum amount of flow from a specified source node s to another specified sink node t in a network with arc capacities u_{ij}'s. If we interpret u_{ij} as the maximum flow rate of arc (i, j), then the maximum flow problem identifies the maximum steady-state flow that the network can send from node s to node t per unit time. We can formulate this problem as a minimum cost flow problem in the following manner. We set $b(i) = 0$ for all $i \in N$, $l_{ij} = c_{ij} = 0$ for all $(i, j) \in A$, and introduce an additional arc (t, s) with cost $c_{ts} = -1$ and flow bounds $l_{ts} = 0$ and capacity $u_{ts} = \infty$. Then the minimum cost flow solution maximizes the flow on arc (t, s); but since any flow on arc (t, s) must travel from node s to node t through the arcs in A (since each $b(i) = 0$), the solution to the minimum cost flow problem will maximize the flow from node s to node t in the original network.

Some applications impose nonzero lower bounds l_{ij} as well as capacities u_{ij} on the arc flows. We refer to this problem as the *maximum flow problem with lower bounds*.

The maximum flow problem is very closely related to another fundamental network optimization problem, known as the *minimum cut problem*. Recall from Section 2 that an s–t cut is a set of arcs whose deletion from the network creates a disconnected network with two components, one component S contains the source node s and the other component $\overline{S}$ contains the sink node t. The capacity of an s–t cut $[S, \overline{S}]$ is the sum of the capacities of all arcs (i, j) with $i \in S$ and $j \in \overline{S}$. The minimum cut problem seeks an s–t cut of minimum capacity.

The *max-flow min-cut theorem*, a celebrated theorem in network optimization, establishes a relationship between the maximum flow problem and the minimum cut problem, namely, the value of the maximum flow from the source node s to the sink node t equals the capacity of a minimum s–t cut. This theorem allows us to determine in $O(m)$ time a minimum cut from the optimal solution of the maximum flow problem.

The maximum flow problem arises in a wide variety of situations and in several forms. Examples of the maximum flow problem include determining the maximum steady state flow of (i) petroleum products in a pipeline network, (ii) cars in a road network; (iii) messages in a telecommunication network; and (iv) electricity in an electrical network. Sometimes the maximum flow problem occurs as a subproblem in the solution of more difficult network problems such as the minimum cost flow problem or the generalized flow problem. The maximum flow problem also arises

in a number of combinatorial applications that on the surface might not appear to be maximum flow problems at all. In this section, we describe a few such applications.

Application 6. Matrix rounding problem [Bacharach, 1966]

This application is concerned with consistent rounding of the elements, row sums, and column sums of a matrix. We are given a $p \times q$ matrix of *real* numbers $D = \{d_{ij}\}$, with row sums α_i and column sums β_j. At our discretion, we can round any real number α to the next smaller integer $\lfloor \alpha \rfloor$ or to the next larger integer $\lceil \alpha \rceil$. The matrix rounding problem requires that we round the matrix elements, and the row and column sums of the matrix so that the sum of the rounded elements in each row equals the rounded row sum, and the sum of the rounded elements in each column equals the rounded column sum. We refer to such a rounding as a *consistent rounding*.

This matrix rounding problem arises is several application contexts. For example, the U.S. Census Bureau uses census information to construct millions of tables for a wide variety of purposes. By law, the bureau has an obligation to protect the source of its information and not disclose statistics that could be attributed to any particular individual. We might disguise the information in a table as follows. We round off each entry in the table, including the row and column sums, either up or down to a multiple of a constant k (for some suitable value of k), so that the entries in the table continue to add to the (rounded) row and column sums, and the overall sum of the entries in the new table adds to a rounded version of the overall sums in the original table. This Census Bureau problem is the same as the matrix rounding problem discussed earlier except that we need to round each element to a multiple of $k \geq 1$ instead of rounding it to a multiple of 1. For the moment, let us suppose that $k = 1$ so that this application is a matrix rounding problem in which we round any element to a nearest integer.

We shall formulate this problem and some of the subsequent applications as a problem known as the *feasible flow problem*. In the feasible flow problem, we wish to determine a flow x in a network $G = (N, A)$ satisfying the following constraints:

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i), \quad \text{for } i \in N, \tag{8a}$$

$$0 \leq x_{ij} \leq u_{ij}, \qquad \text{for all } (i, j) \in A. \tag{8b}$$

As we noted in Section 2, this model includes feasible flow applications with nonzero lower bounds on arc flows since we can transform any such problem into an equivalent problem with zero lower bounds on the arc flows.

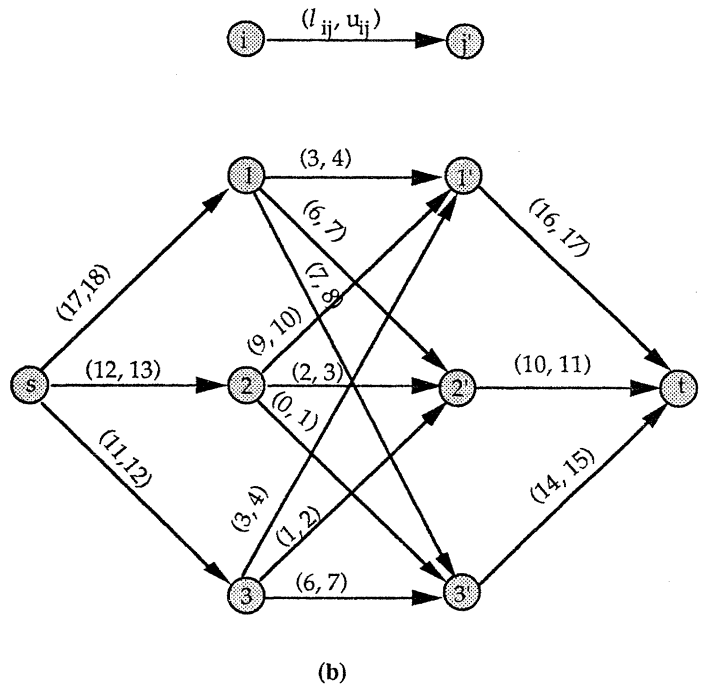
We assume that $\sum_{i\in N} b(i) = 0$. We can solve the feasible flow problem by solving a maximum flow problem defined on an augmented network as follows. We introduce two new nodes, a source node s and a sink node t. For each node i with $b(i) > 0$, we add an arc (s, i) with capacity $b(i)$, and for each node i with $b(i) < 0$, we add an arc (i, t) with capacity $-b(i)$. We refer to the new network as

the *transformed network*. Then we solve a maximum flow problem from node s to node t in the transformed network. It is easy to show that the problem (8) has a feasible solution if and only if the maximum flow saturates all the arcs emanating from the source node.

We show how we can discover such a rounding scheme by solving a feasible flow problem for a network with nonnegative lower bounds on arc flows. Figure 7b shows the maximum flow network for the matrix rounding data shown in Figure 7a. This network contains a node i corresponding to each row i and a node j' corresponding to each column j. Observe that this network contains an arc (i, j') for each matrix element d_{ij}, an arc (s, i) for each row sum, and an arc (j', t) for each column sum. The lower and upper bounds of arc (k, l) corresponding to the matrix element, row sum, or column sum of value a are $\lfloor \alpha \rfloor$ and $\lceil \alpha \rceil$, respectively. It is easy to establish a one-to-one correspondence between the consistent roundings of the matrix and feasible integral flows in the associated

				row sum
	3.1	6.8	7.3	17.2
	9.6	2.4	0.7	12.7
	3.6	1.2	6.5	11.3
column sum	16.3	10.4	14.5	

(a)



(b)

Fig. 7. (a) Matrix rounding problem. (b) Associated network.

network. We know that there is a feasible integral flow since the original matrix elements induce a feasible fractional flow, and maximum flow algorithms produce all integer flows. Consequently, we can find a consistent rounding by solving a maximum flow problem with lower bounds.

The solution of a matrix rounding problem in which we round every element to a multiple of some positive integer k, as in the Census application we mentioned previously, is similar. In this case, we define the associated network as before, but now defining the lower and upper bounds for any arc with an associated real number a as the greatest multiple of k less than or equal to a and the smallest multiple of k greater than or equal to a.

Application 7. Baseball elimination problem [Schwartz, 1966]

At a particular point in the baseball season, each of $n+1$ teams in the American League, which we number as 0, 1, ..., n, has played several games. Suppose that team i has won w_i of the games it has already played and that g_{ij} is the number of games that teams i and j have yet to play with each other. No game ends in a tie. An avid and optimistic fan of one of the teams, the Boston Red Sox, wishes to know if his team still has a chance to win the league title. We say that we can *eliminate* a specific team 0, the Red Sox, if for every possible outcome of the unplayed games, at least one team will have more wins than the Red Sox. Let $w_{\max}$ denote w_0 (i.e., the number of wins of team 0) plus the total number of games team 0 has yet to play, which, in the best of all possible worlds, is the number of victories the Red Sox can achieve. Then, we cannot eliminate team 0 if in some outcome of the remaining games to be played throughout the league, $w_{\max}$ is at least as large as the possible victories of every other team. We want to determine whether we can or cannot eliminate team 0.

We can transform this baseball elimination problem into a feasible flow problem on a bipartite network shown in Figure 8, whose node set is $N_1 \cup N_2$. The node set for this network contains (i) a set N_1 of n *team nodes* indexed 1 through n, (ii) $n(n-1)/2$ *game nodes* of the type i–j for each $1 \le i \le j \le n$, and (iii) a *source node* s. Each game node i–j has a demand of g_{ij} units and has two incoming arcs $(i, i$–$j)$ and $(j, i$–$j)$ The flows on these two arcs represent the number of victories for team i and team j, respectively, among the additional g_{ij} games that these two teams have yet to play against each other (which is the required flow into the game node i–j). The flow x_{si} on the source arc (s, i) represents the total number of additional games that team i wins. We cannot eliminate team 0 if this network contains a feasible flow x satisfying the conditions

$$w_{\max} \ge w_i + x_{si}, \quad \text{for all } i = 1, \ldots, n,$$

which we can rewrite as

$$x_{si} \le w_{\max} - w_i, \quad \text{for all } i = 1, \ldots, n.$$

This observation explains the capacities of arcs shown in the figure. We have thus shown that if the feasible flow problem shown in Figure 8 admits a feasible

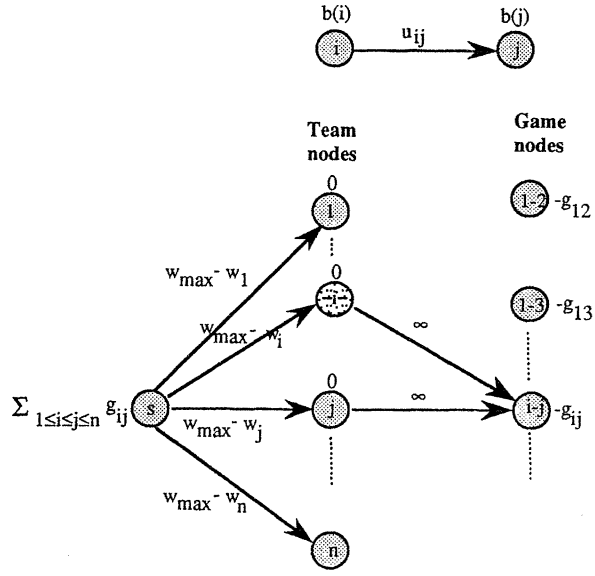


Fig. 8. Network formulation of the baseball elimination problem.

flow, then we cannot eliminate team 0; otherwise, we can eliminate this team and our avid fan can turn his attention to other matters.

Application 8. Distributed computing on a two-processor computer [Stone, 1977]

This application concerns assigning different modules (subroutines) of a program to two processors of a computer system in a way that minimizes the collective costs of interprocessor communication and computation. The two processors need not be identical. We wish to execute a large program consisting of several modules that interact with each other during the program's execution. The cost of executing each module on the two processors is known in advance and might vary from one processor to the other because of differences in the processors' memory, control, speed, and arithmetic capabilities. Let a_i and b_i denote the cost of computation of module i on processors 1 and 2, respectively. Assigning different modules to different processors incurs relatively high overhead costs due to interprocessor communication. Let c_{ij} denote the interprocessor communication cost if modules i and j are assigned to different processors; we do not incur this cost if we assign modules i and j to the same processor. The cost structure might suggest that we allocate two modules to different processors – we need to balance this cost against the communication costs that we incur by allocating the jobs to different processors. Therefore, we wish to allocate modules of the program on the two processors so that we minimize the total processing and interprocessor communication cost.

To formulate this problem as a minimum cut problem on an undirected network, we define a source node s representing processor 1, a sink node t representing processor 2, and a node for every module of the program. For every node i, other than the source and sink nodes, we include an arc (s, i) of capacity β_i and an

i	1	2	3	4
α_i	6	5	10	4
β_i	4	10	3	8

(a)

$\{c_{ij}\} =$

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

(b)

Fig. 9. Data for the distributed computing model.

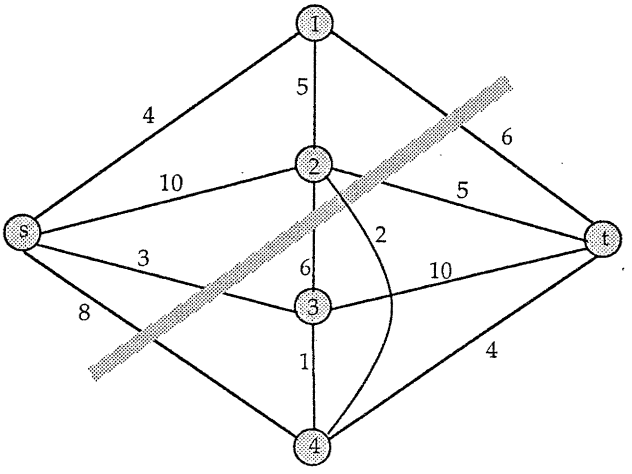


Fig. 10. Network for the distributed computing model.

arc (i, t) of capacity α_i. Finally, if module i interacts with module j during the program's execution, we include arc (i, j) with a capacity equal to c_{ij}. Figures 9 and 10 give an example of this construction. Figure 9 gives the data of this problem and Figure 10 gives the corresponding network.

We now observe a one-to-one correspondence between s–t cuts in the network and assignments of modules to the two processors; moreover, the capacity of a cut equals the cost of the corresponding assignment. To establish this result, let A_1 and A_2 be an assignment of modules to processors 1 and 2, respectively. The cost of this assignment is $\sum_{i \in A_1} a_i + \sum_{i \in A_2} b_i + \sum_{(i,j) \in A_1 \times A_2} c_{ij}$. The s–t cut corresponding to this assignment is $(\{s\} \cup A_1, \{t\} \cup A_2)$. The approach we used to construct the network implies that this cut contains an arc (i, t) of capacity α_i for every $i \in A_1$, an arc (s, i) of capacity β_i for every $i \in A_2$, and all arcs (i, j) with $i \in A_1$ and $j \in A_2$ with capacity c_{ij}. The cost of the assignment A_1 and A_2 equals the capacity of the cut $(\{s\} \cup A_1, \{t\} \cup A_2)$. (The reader might wish to verify this conclusion on the example given in Figure 10 with $A_1 = \{1, 2\}$ and $A_2 = \{3, 4\}$.) Consequently, the minimum s–t cut in the network gives the minimum cost assignment of the modules to the two processors.

Application 9. Scheduling on uniform parallel machines [Federgruen & Groenevelt, 1986]

In this application, we consider the problem of scheduling a set J of jobs on M uniform parallel machines. Each job $j \in J$ has a processing requirement p_j (denoting the number of machine days required to complete the job); a release date r_j (representing the beginning of the day when job j becomes available for processing); and a due date $d_j \geq r_j + p_j$ (representing the beginning of the day by which the job must be completed). We assume that a machine can work on only one job at a time and that each job can be processed by at most one machine at a time. However, we allow *preemptions*, i.e., we can interrupt a job and process it on different machines on different days. The scheduling problem is to determine a feasible schedule that completes all jobs before their due dates or to show that no such schedule exists.

This type of preemptive scheduling problem arises in batch processing systems when each batch consists of a large number of units. The feasible scheduling problem, described in the preceding paragraph, is a fundamental problem in this situation and can be used as a subroutine for more general scheduling problems, such as the maximum lateness problem, the (weighted) minimum completion time problem, and the (weighted) maximum utilization problem.

To illustrate the formulation of the feasible scheduling problem as a maximum flow problem, we use the scheduling data described in Figure 11.

First, we rank all the release and due dates, r_j and d_j for all j, in ascending order and determine $P \leq 2|J| - 1$ mutually disjoint intervals of dates between consecutive milestones. Let $T_{k,l}$ denote the interval that starts at the beginning of date k and ends at the beginning of date $l + 1$. For our example, this order of release and due dates is 1, 3, 4, 5, 7, 9. We have five intervals represented by $T_{1,2}$, $T_{3,3}$, $T_{4,4}$, $T_{5,6}$ and $T_{7,8}$. Notice that within each interval $T_{k,l}$, the set of available jobs (that is, those released, but not yet due) does not change: we can process all jobs j with $r_j \leq k$ and $d_j \geq l + 1$ in the interval.

We formulate the scheduling problem as a maximum flow problem on a bipartite network G as follows. We introduce a source node s, a sink node t, a node corresponding to each job j, and a node corresponding to each interval $T_{k,l}$, as shown in Figure 12. We connect the source node to every job node j with an arc with capacity p_j, indicating that we need to assign a minimum of p_j machine

Job (j)	1	2	3	4
Processing time (p_j)	1.5	1.25	2.1	3.6
Release time (r_j)	3	1	3	5
Due date (d_j)	5	4	7	9

Fig. 11. A scheduling problem.

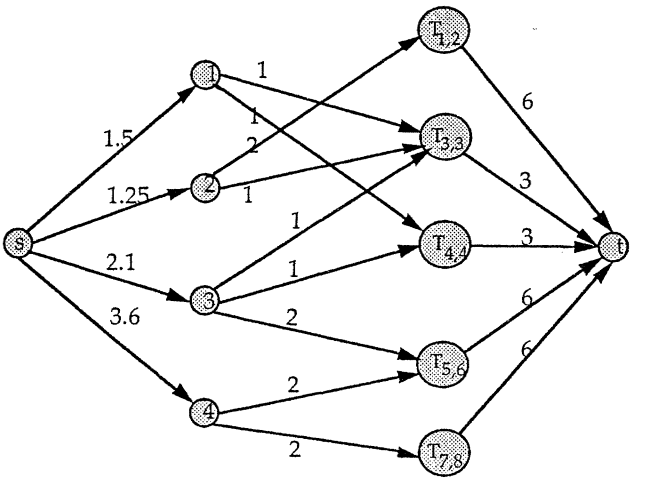


Fig. 12. Network for scheduling uniform parallel machines.

days to job j. We connect each interval node $T_{k,l}$ to the sink node t by an arc with capacity $(l - k + 1)M$, representing the total number of machine days available on the days from k to l. Finally, we connect job node j to every interval node T_I if $r_j \leq k$ and $d_j \geq l + 1$ by an arc with capacity $(l - k + 1)$ which represents the maximum number of machines that we can allot to job j on the days from k to l. We next solve a maximum flow problem on this network: the scheduling problem has a feasible schedule if and only if the maximum flow value equals $\sum_{j \in J} p_j$ (alternatively, for every node j, the flow on arc (s, j) is p_j). The validity of this formulation is easy to establish by showing a one-to-one correspondence between feasible schedules and flows of value equal to $\sum_{j \in J} p_j$ from the source to the sink.

Application 10. Tanker scheduling [Dantzig & Fulkerson, 1954]

A steamship company has contracted to deliver perishable goods between several different origin–destination pairs. Since the cargo is perishable, the customers have specified precise dates (i.e., delivery dates) when the shipments must reach their destinations. (The cargoes may not arrive early or late.) The steamship company wants to determine the minimum number of ships needed to meet the delivery dates of the shiploads.

To illustrate a modeling approach for this problem, we consider an example with four shipments; each shipment is a full shipload with the characteristics shown in Figure 13a. For example, as specified by the first row in this figure, the company must deliver one shipload available at port A and destined for port C on day 3. Figures 13b and 13c show the transit times for the shipments (including allowances for loading and unloading the ships) and the return times (without a cargo) between the ports.

We solve this problem by constructing a network shown in Figure 14a. This network contains a node for each shipment and an arc from node i to node j if it

Shipment	Origin	Destination	Delivery date
1	Port A	Port C	3
2	Port A	Port C	8
3	Port B	Port D	3
4	Port B	Port C	6

(a)

	C	D
A	3	2
B	2	3

(b)

	A	B
C	2	1
D	1	2

(c)

Fig. 13. Data for the tanker scheduling problem.

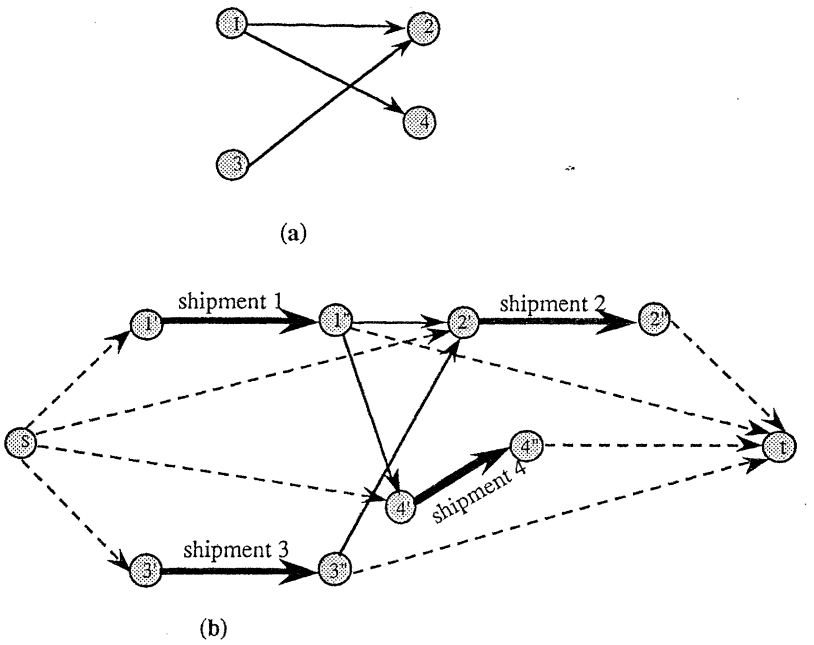


Fig. 14. Network formulation of the tanker scheduling problem. (a) Network of feasible sequences of two consecutive shipments. (b) Maximum flow model.

is possible to deliver shipment j after completing shipment i; that is, the start time of shipment j is no earlier than the delivery time of shipment i plus the travel time from the destination of shipment i to the origin of shipment j. A directed path in this network corresponds to a feasible sequence of shipment pickups and deliveries. The tanker scheduling problem requires that we identify the minimum number of directed paths that will contain each node in the network on exactly one path.

We can transform this problem into the framework of the maximum flow problem as follows. We split each node i into two nodes i' and i'' and add the arc (i', i''). We set the lower bound on each arc (i', i''), called the *shipment arc*, equal to one so that at least unit flow passes through this arc. We also add a source node s and connect it to the origin of each shipment (to represent putting a ship

into service), and we add a sink node t and connect each destination node to it (to represent taking a ship out of service). We set the capacity of each arc in the network to value one. Figure 14b shows the resulting network for our example. In this network, each directed path from the source node s to the sink node t corresponds to a feasible schedule for a single ship. As a result, a feasible flow of value v in this network decomposes into schedules of v ships, and our problem reduces to identifying a feasible flow of minimum value. We note that the zero flow is not feasible because shipment arcs have unit lower bounds. We can solve this problem, which is known as the *minimum value problem*, in the following manner. We first remove lower bounds on the arcs using the transformation described in Section 2. We then establish a feasible flow in the network by solving a maximum flow problem, as described in Application 6. Although the feasible flow x satisfies all of the constraints, the amount of flow sent from node s to node t might exceed the minimum. In order to find a minimum flow, we need to return the maximum amount of flow from t to s. To do so, we find a maximum flow from node t to node s in the residual network, which is defined as follows: For each arc (i, j) in the original network, the residual network contains an arc (i, j) with capacity $u_{ij} - x_{ij}$, and another arc (j, i) with capacity $x_{ij} - l_{ij}$. A maximum flow between nodes t to s in the residual network corresponds to returning a maximum amount of flow from node t to node s, and provides optimal solution of the minimum value problem.

As exemplified by this application, finding a minimum (or maximum) flow in the presence of lower bounds on arc flows typically requires solving two maximum flow problems. The solution to the first maximum flow problem is a feasible flow. The solution to the second maximum flow problem establishes a minimum (or maximum) flow.

Several other applications that bear a close resemblance to the tanker scheduling problem are solvable using the same technique. We next briefly introduce some of these applications.

Optimal coverage of sporting events. A group of reporters wants to cover a set of sporting events in an Olympiad. The sports events are held in several stadiums throughout a city. We know the starting time of each event, its duration, and the stadium where it is held. We are also given the travel times between different stadiums. We want to determine the least number of reporters required to cover the sporting events.

Bus scheduling problem. A mass transit company has p routes that it wishes to service using the fewest possible number of buses. To do so, it must determine the most efficient way to combine these routes into bus schedules. The starting time for route i is a_i and the finishing time is b_i. The bus requires r_{ij} time to travel from the point of destination of route i to the point of origin of route j.

Machine setup problem. A job shop needs to perform eight tasks on a particular day. We know the start and end times of each task. The workers must perform these tasks according to this schedule and so that exactly one worker performs

each task. A worker cannot work on two jobs at the same time. We also know the setup time (in minutes) required for a worker to go from one task to another. We wish to find the minimum number of workers to perform the tasks.

Additional applications

The maximum flow problem arises in many other applications, including: (1) *problem of representatives* [Hall, 1956]; (2) *open pit mining* [Johnson, 1968]; (3) *the tournament problem* [Ford & Johnson, 1959]; (4) *police patrol problem* [Khan, 1979]; (5) *nurse staff scheduling* [Khan & Lewis, 1987]; (6) *solving a system of equations* [Lin, 1986]; (7) *statistical security of data* [Gusfield, 1988; Kelly, Golden & Assad, 1992]; (8) *minimax transportation problem* [Ahuja, 1986]; (9) *network reliability testing* [Van Slyke & Frank, 1972]; (10) *maximum dynamic flows* [Ford & Fulkerson, 1958]; (11) *preemptive scheduling on machines with different speeds* [Martel, 1982]; (12) *multifacility rectilinear distance location problem* [Picard & Ratliff, 1978]; (13) *selecting freight handling terminals* [Rhys, 1970]; (14) *optimal destruction of military targets* [Orlin, 1987]; and (15) *fly away kit problem* [Mamer & Smith, 1982]. The following papers describe additional applications of the maximum flow problem or provide additional references: Berge & Ghouila-Houri [1962]; McGinnis & Nuttle [1978], Picard & Queyranne [1982], Abdallaoui [1987], Gusfield, Martel & Fernandez-Baca [1987], Gusfield & Martel [1992], and Gallo, Grigoriadis & Tarjan [1989].

5. Minimum cost flows

The minimum cost flow model is the most fundamental of all network flow problems. In this problem, as described in Section 1, we wish to determine a least cost shipment of a commodity through a network that will satisfy demands at certain nodes from available supplies at other nodes. This model has a number of familiar applications: the distribution of a product from manufacturing plants to warehouses, or from warehouses to retailers; the flow of raw material and intermediate goods through the various machining stations in a production line; the routing of automobiles through an urban street network; and the routing of calls through the telephone system. Minimum cost flow problems arise in almost all industries, including agriculture, communications, defense, education, energy, health care, manufacturing, medicine, retailing, and transportation. Indeed, minimum cost flow problems are pervasive in practice. In this section, by considering a few selected applications, we illustrate some of these possible uses of minimum cost flow problems.

Application 11. Leveling mountainous terrain [Farley, 1980]

This application was inspired by a common problem facing civil engineers when they are building road networks through hilly or mountainous terrain. The

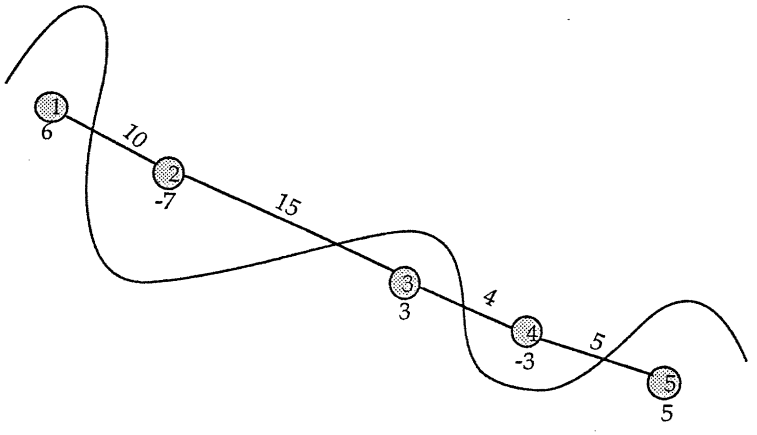


Fig. 15. A portion of the terrain graph.

problem concerns the distribution of earth from high points to low points of the terrain in order to produce a leveled road bed. The engineer must determine a plan for leveling the route by specifying the number of truck loads of earth to move between various locations along the proposed road network.

We first construct a *terrain graph* which is an undirected graph whose nodes represent locations with a demand for earth (low points) or locations with a supply of earth (high points). An arc of this graph represents an available route for distributing the earth and the cost of this arc represents the cost of per truck load of moving earth between the two points. (A *truckload* is the basic unit for redistributing the earth.) Figure 15 shows a portion of the terrain graph.

A leveling plan for a terrain graph is a flow (set of truckloads) that meets the demands at nodes (levels the low points) by the available supplies (by earth obtained from high points) at the minimum cost (for the truck movements). This model is clearly a minimum cost flow problem in the terrain graph.

Application 12. Reconstructing the left ventricle from X-ray projections [Slump & Gerbrands, 1982]

This application describes a minimum cost flow model for reconstructing the three-dimensional shape of the left ventricle from biplane angiocardiograms that the medical profession uses to diagnose heart diseases. To conduct this analysis, we first reduce the three-dimensional reconstruction problem into several two-dimensional problems by dividing the ventricle into a stack of parallel cross sections. Each two-dimensional cross section consists of one connected region of the left ventricle. During a cardiac catheterization, doctors inject a dye known as Roentgen contrast agent into the ventricle; by taking X-rays of the dye, they would like to determine what portion of the left ventricle is functioning properly (that is, permitting the flow of blood). Conventional biplane X-ray installations do not permit doctors to obtain a complete picture of the left ventricle; rather, these

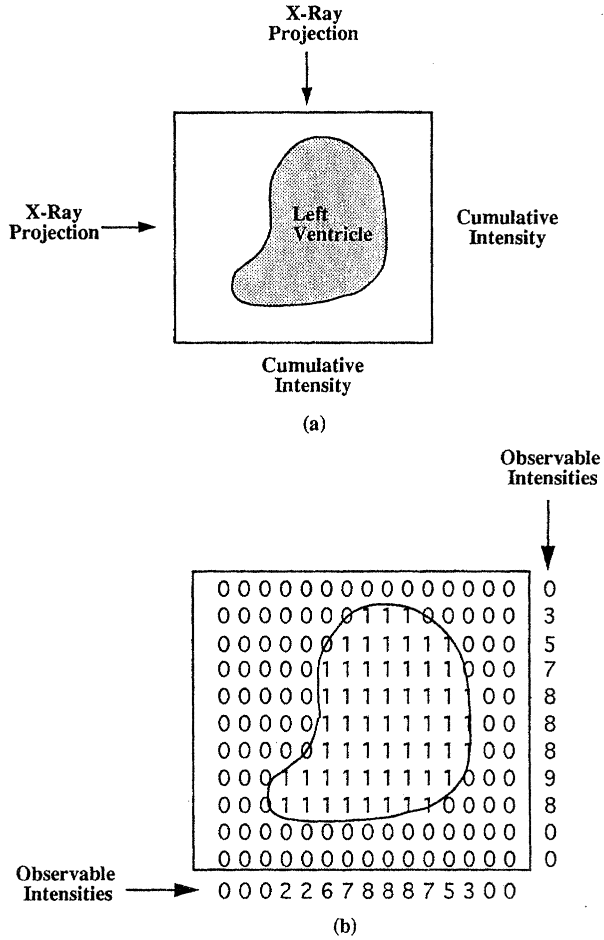


Fig. 16. Using X-Ray projections to measure a left ventricle.

X-rays provide one-dimensional projections that record the total intensity of the dye along two axes (see Figure 16). The problem is to determine the distribution of the cloud of dye within the left ventricle, and thus the shape of the functioning portion of the ventricle, assuming that the dye mixes completely with the blood and fills the portions that are functioning properly.

We can conceive of a cross section of the ventricle as a $p \times r$ binary matrix: a 1 in a position indicates that the corresponding segment of the ventricle allows blood to flow and a 0 indicates that it doesn't permit blood to flow. The angiocardiograms gives the cumulative intensity of the contrast agent in two planes which we can translate into row and column sums of the binary matrix. The problem is then to construct the binary matrix given its row and column sums. This problem is a special case of the feasible flow problem (discussed in Application 6) on a network with a node i for each row i of the matrix with the supply equal to the cumulative intensity of the row, a node j' for each column j of the matrix with demand equal to the cumulative intensity of the column, and a unit capacity arc from each row node i to each column node j'.

Typically, the number of feasible solutions are quite large; and, these solutions might differ substantially. To constrain the feasible solutions, we might use certain facts from our experience that indicate that a solution is more likely to contain certain segments rather than others. Alternatively, we can use *a priori* information: for example, after some small time interval, the cross sections might resemble cross sections determined in a previous examination. Consequently, we might attach a probability p_{ij} that a solution will contain an element (i, j) of the binary matrix and might want to find a feasible solution with the largest possible cumulative probability. This problem is equivalent to a minimum cost flow problem on the network we have already described.

Application 13. Optimal loading of a hopping airplane [Gupta, 1985; Lawania, 1990]

A small commuter airline uses a plane, with a capacity to carry at most p passengers, on a 'hopping flight' as shown in Figure 17a. The hopping flight visits the cities 1, 2, 3, ..., n, in a fixed sequence. The plane can pick up passengers at any node and drop them off at any other node. Let b_{ij} denote the number of passengers available at node i who want to go to node j, and let f_{ij} denote the fare per passenger from node i to node j. The airline would like to determine the number of passengers that the plane should carry between the various origins to destinations in order to maximize the total fare per trip while never exceeding the plane capacity.

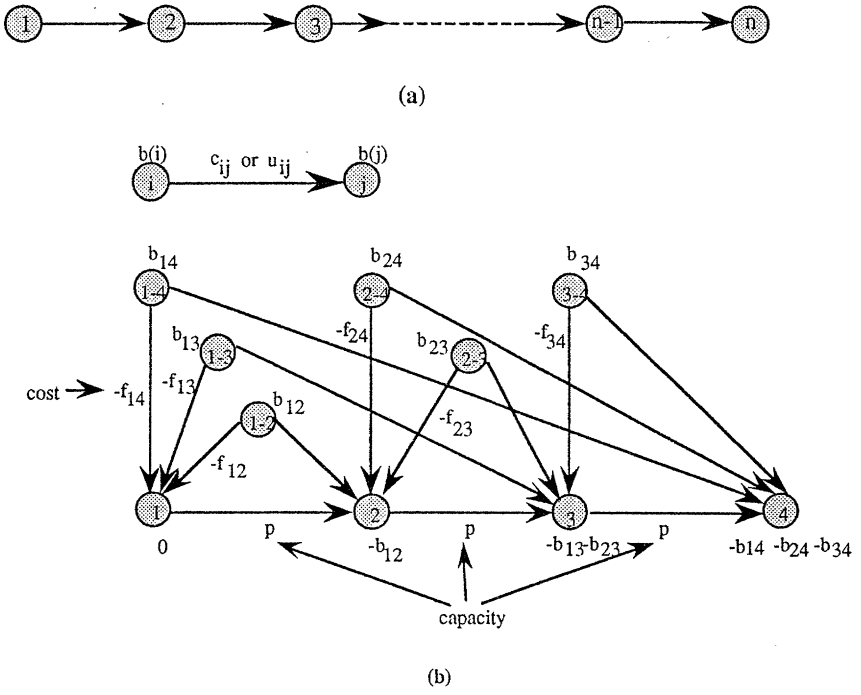


Fig. 17. Formulating the hopping plane flight problem as a minimum cost flow problem.

Figure 17b shows a minimum cost flow formulation of this hopping plane flight problem. The network contains data for only those arcs with nonzero costs and with finite capacities: any arc without an associated cost has a zero cost; any arc without an associated capacity has an infinite capacity. Consider, for example, node 1. Three types of passengers are available at node 1, those whose destination is node 2, node 3, or node 4. We represent these three types of passengers by the nodes 1-2, 1-3, and 1-4 with supplies b_{12}, b_{13}, and b_{14}. A passenger available at any such node, say 1-3, either boards the plane at its origin node by flowing through the arc (1-3, 1), and thus incurring a cost of $-f_{13}$ units, or never boards the plane which we represent by the flow through the arc (1-3, 3). Therefore, each loading of the plane has a corresponding feasible flow of the same cost in the network. The converse result is true (and is easy to establish): each feasible flow in the network has a corresponding feasible airplane loading. Thus, this formulation correctly models the hopping plane application.

Application 14. Directed Chinese postman problem [Edmonds & Johnson, 1973]

Leaving from his home post office, a postman needs to visit the households on each block in his route, delivering and collecting letters, and then returning to the post office. He would like to cover this route by travelling the minimum possible distance. Mathematically, this problem has the following form: Given a network $G = (N, A)$ whose arcs (i, j) have an associated nonnegative length c_{ij}, we wish to identify a walk of minimum length that starts at some node (the post office), visits each arc of the network at least once, and returns to the starting node. This problem has become known as the *Chinese postman problem* because it was first discussed by a Chinese mathematician, K. Mei-Ko. The Chinese postman problem arises in other application settings as well; for instance, in patrolling streets by a police officer, routing of street sweepers and household refuse collection vehicles, fuel oil delivery to households, and the spraying of roads with sand during snow storms. In this application, we discuss the Chinese postman problem on directed networks.

In the Chinese postman problem on directed networks, we are interested in a closed (directed) walk that traverses each arc of the network at least once. The network need not contain any such walk! Figure 18 shows an example. The network contains the desired walk if and only if every node in the network is reachable from every other node; that is, it is *strongly connected*. The strong connectivity

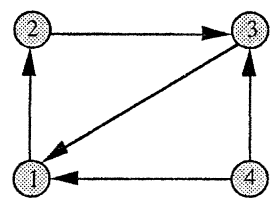


Fig. 18. Network containing no feasible solution for the Chinese postman problem.

of a network can be easily determined in $O(m)$ time [see, e.g., Ahuja, Magnanti & Orlin, 1993]. We shall henceforth assume that the network is strongly connected.

In an optimal walk, a postman might traverse arcs more than once. The minimum length walk minimizes the sum of lengths of the arcs that the walk repeats. Let x_{ij} denote the number of times the postman traverses an arc (i, j) in a walk. Any walk of the postman must satisfy the following conditions:

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0, \qquad \text{for all } i \in N, \tag{9a}$$

$$x_{ij} \geq 1, \qquad \text{for all } (i, j) \in A. \tag{9b}$$

The constraints (9a) state that the postman enters a node the same number of times that he/she leaves it. The constraints (9b) state that the postman must visit each arc at least once. Any solution x satisfying (9a) and (9b) defines a postman's walk. We can construct such a walk in the following manner. We replace each arc (i, j) with flow x_{ij} with x_{ij} copies of the arc, each carrying a unit flow. Let A' denote the resulting arc set. Since the outflow equals inflow for each node in the flow x, once we have transformed the network, the outdegree of each node will equal its indegree. This implies that we can decompose the arc set A' into a set of at most m directed cycles. We can connect these cycles together to form a closed walk as follows. The postman starts at some node in one of the cycles, say W_1, and visits the nodes (and arcs) of W_1 in order until he either returns to the node he started from, or encounters a node that also lies in a directed cycle not yet visited, say W_2. In the former case, the walk is complete; and in the latter case, the postman visits cycle W_2 first before resuming his visit of the nodes in W_1. While visiting nodes in W_2, the postman follows the same policy, i.e., if he encounters a node lying on another directed cycle W_3 not yet visited, then he visits W_3 first before visiting the remaining nodes in W_2, and so on. We illustrate this method on a numerical example. Let A' be as indicated in Figure 19a. This solution decomposes into three directed cycles W_1, W_2 and W_3. As shown in Figure 19b, the postman starts at node a and visits the nodes in the following order: a–b–d–g–h–c–d–e–b–c–f–a.

This discussion shows that the solution x defined by a feasible walk for the postman satisfies (9), and, conversely, every feasible solution of (9) defines a walk of the postman. The length of a walk equals $\sum_{(i,j)\in A} c_{ij}x_{ij}$. Therefore, the Chinese postman problem seeks a solution x that minimizes $\sum_{(i,j)\in A} c_{ij}x_{ij}$, subject to the set of constraints (9). This problem is clearly an instance of the minimum cost flow problem.

Application 15. Racial balancing of schools [Belford & Ratliff, 1972]

Suppose a school district has S schools. For the purpose of this formulation, we divide the school district into L district locations and let b_i and w_i denote the number of black and white students at location i. These locations might, for example, be census tracts, bus stops, or city blocks. The only restrictions on the

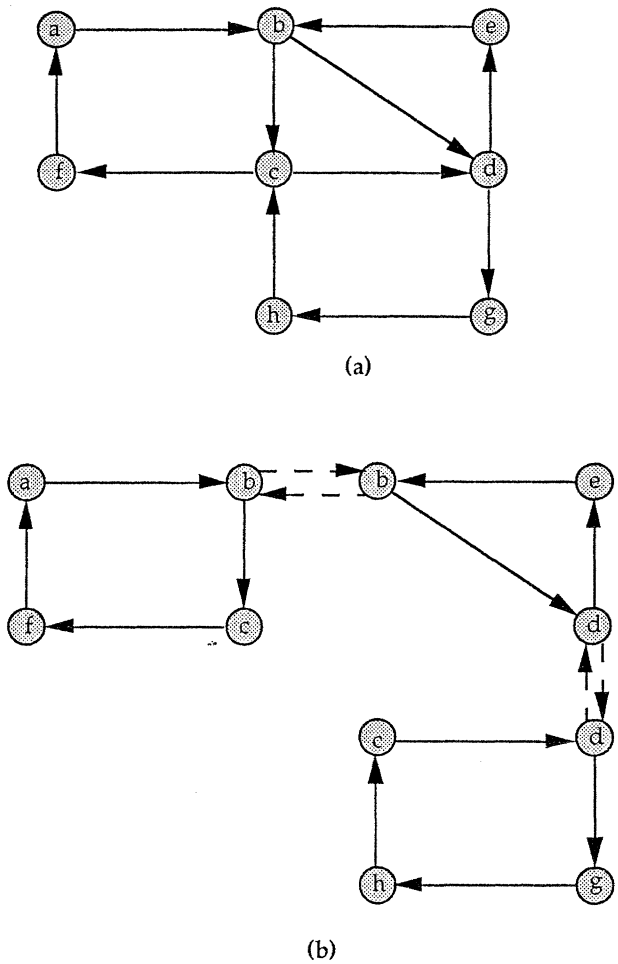


Fig. 19. Constructing a closed walk for the postman.

locations is that they be finite in number and that there be a single distance measure d_{ij} that reasonably approximates the distance any student at location i must travel if he or she is assigned to school j. We make the reasonable assumption that we can compute the distances d_{ij} before assigning students to schools. School j can enroll u_j students. Finally, let $\underline{p}$ denote a lower bound and $\overline{p}$ denote an upper bound on the percentage of blacks assigned to each school (we choose these numbers so that school j has same percentage of blacks as does the school district). The objective is to assign students to schools in a manner that maintains the stated racial balance and minimizes the total distance travelled by the students.

We model this problem as a minimum cost flow problem. Figure 20 shows the minimum cost flow network for a three-location, two-school problem. Rather than describe the general model formally, we will merely describe the model ingredients for this figure. In this formulation, we model each location i as two nodes l_i' and l_i'' and each school j as two nodes s_j' and s_j''. The decision variables for this problem

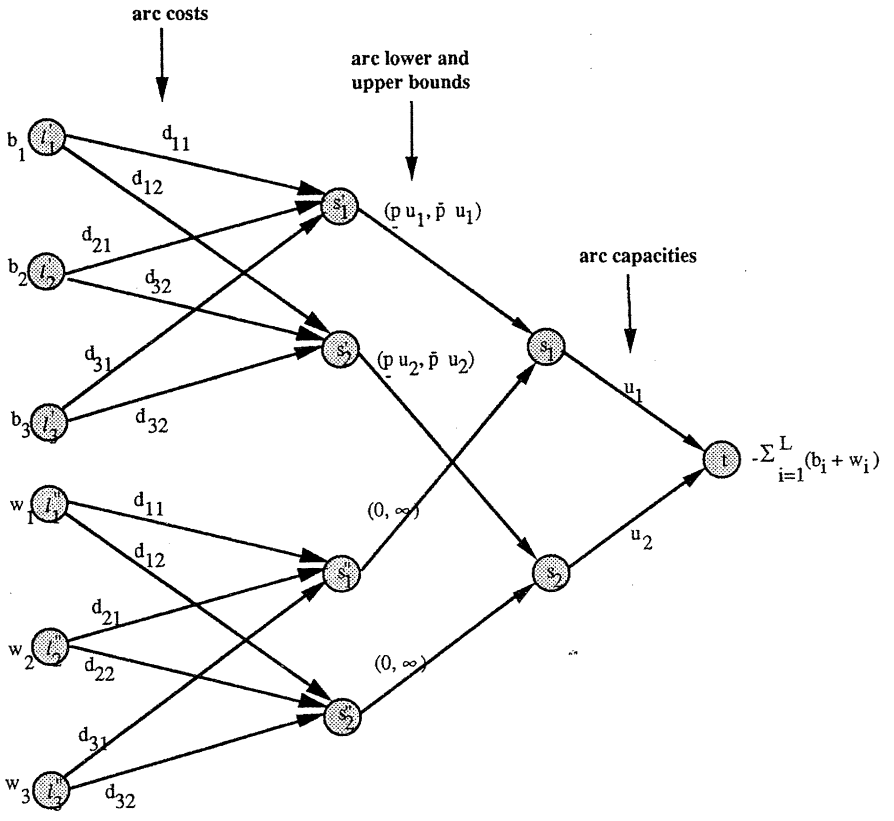


Fig. 20. Network for the racial balancing of schools.

are the number of black students assigned from location i to school j (which we represent by an arc from node l_i' to node s_j') and the number of white students assigned from location i to school j (which we represent by an arc from node l_i'' to node s_j''). These arcs are uncapacitated and we set their per unit flow cost equal to d_{ij}. For each j, we connect the nodes s_j' and s_j'' to the school node s_j. The flow on the arcs (s_j', s_j) and (s_j'', s_j) denotes the total number of black and white students assigned to school j. Since each school must satisfy lower and upper bounds on the number of black students it enrolls, we set the lower and upper bounds of the arc (s_j', s_j) equal to $(\underline{p}u_j, \bar{p}u_j)$. Finally, we must satisfy the constraint that school j enrolls at most u_j students. We incorporate this constraint in the model by introducing a sink node t and joining each school node j to node t by an arc of capacity u_j. As is easy to verify, this minimum cost flow problem correctly models the racial balancing application.

Additional applications

A complete list of additional applications of the minimum cost flow problem is too vast to mention here. A partial list of additional references is: (1) *distribution problems* [Glover & Klingman, 1976]; (2) *building evacuation models* [Chalmet,

Francis & Saunders, 1982]; (3) *scheduling with consecutive ones in columns* [Veinott & Wagner, 1962]; (4) *linear programs with consecutive circular ones in rows* [Bartholdi, Ratliff & Orlin, 1980]; (5) *the entrepreneur's problem* [Prager, 1957]; (6) *optimal storage policy for libraries* [Evans, 1984]; (7) *zoned warehousing* [Evans, 1984]; (8) *allocation of contractors to public works* (Cheshire, McKinnon & Williams, 1984]; (9) *phasing out capital equipment* [Daniel, 1973]; (10) *the terminal assignment problem* [Esau & Williams, 1966]; (11) *capacitated maximum spanning trees* [Garey & Johnson, 1979]; (12) *caterer problem* [Jacobs, 1954]; (13) *allocating receivers to transmitters* [Dantzig, 1962]; (14) *faculty-course assignment* [Mulvey, 1979]; (15) *automatic karyotyping of chromosomes* [Tso, Kleinschmidt, Mitterreiter & Graham, 1991]; (16) *just-in-time scheduling* [Elmaghraby, 1978; Levner & Nemirovsky, 1991]; (17) *time-cost tradeoff in project management* [Fulkerson, 1961; Kelley, 1961]; (18) *warehouse layout* [Francis & White, 1976]; (19) *rectilinear distance facility location* [Cabot, Francis & Stary, 1970]; (20) *dynamic lot-sizing* [Zangwill, 1969]; (21) *multistage production-inventory planning* [Evans, 1977]; (22) *mold allocation* [Love & Vemuganti, 1978]; (23) *a parking model* [Dirickx & Jennergren, 1975]; (24) *the network interdiction problem* [Fulkerson & Harding, 1977]; (25) *truck scheduling* [Gavish & Schweitzer, 1974]; and (26) *optimal deployment of firefighting companies* [Denardo, Rothblum & Swersey, 1988]; (27) *warehousing and distribution of a seasonal product* [Jewell, 1957]; (28) *economic distribution of coal supplies in the gas industry* [Berrisford, 1960]; (29) *upsets in round robin tournaments* [Fulkerson, 1965]; (30) *optimal container inventory and routing* [Horn, 1971]; (31) *distribution of empty rail containers* [White, 1972]; (32) *optimal defense of a network* [Picard & Ratliff, 1973]; (33) *telephone operator scheduling* [Segal, 1974]; (34) *multifacility minimax location problem with rectilinear distances* [Dearing & Francis, 1974]; (35) *cash management problems* [Srinivasan, 1974]; (36) *multiproduct multifacility production-inventory planning* [Dorsey, Hodgson & Ratliff, 1975]; (37) *'hub' and 'wheel' scheduling problems* [Arisawa & Elmaghraby, 1977]; (38) *warehouse leasing problem* [Lowe, Francis & Reinhardt, 1979]; (39) *multi-attribute marketing models* [Srinivasan, 1979]; (40) *material handling systems* [Maxwell & Wilson, 1981]; (41) *microdata file merging* [Barr & Turner, 1981]; (42) *determining service districts* [Larson & Odoni, 1981]; (43) *control of forest fires* [Kourtz, 1984]; (44) *allocating blood to hospitals from a central blood bank* [Sapountzis, 1984]; (45) *market equilibrium problems* [Dafermos & Nagurney, 1984]; (46) *automatic chromosome classifications* [Tso, 1986]; (47) *city traffic congestion problem* [Zawack & Thompson, 1987]; (48) *satellite scheduling* [Servi, 1989]; (49) *determining k disjoint cuts in a network* [Wagner, 1990]; (50) *controlled rounding of matrices* [Cox & Ernst, 1982]; (51) *scheduling with deferral costs* [Lawler, 1964].

6. The assignment problem

The assignment problem is a special case of the minimum cost flow problem and can be defined as follows: Given a weighted bipartite network $G = (N_1 \cup N_2, A)$

with $|N_1| = |N_2|$ and arc weights c_{ij}, find a one-to-one assignment of nodes in N_1 with nodes in N_2, so that the sum of the costs of arcs in the assignment is minimum. It is easy to notice that if we set $b(i) = 1$ for each node $i \in N_1$ and $b(i) = -1$ for each node $i \in N_2$, then the optimal solution of the minimum cost flow problem in G will be an assignment. For the assignment problem, we allow the network G to be directed or undirected. If the network is directed, then we require that each arc $(i, j) \in A$ has $i \in N_1$ and $j \in N_2$. If the network is undirected, then we make it directed by designating all arcs as pointing from nodes in N_1 to nodes in N_2. We shall, therefore, henceforth assume that G is a directed graph.

Examples of the assignment problem include assigning people to projects, jobs to machines, tenants to apartments, swimmers to events in a swimming meet, and medical school graduates to available internships. We now describe three more clever applications of the assignment problem.

Application 16. Locating objects in space [Brogan, 1989]

This application concerns locating objects in space. To identify an object in (three-dimensional) space, we could use two infrared sensors, located at geographically different sites. Each sensor provides an angle of sight of the object and, hence, the line on which the object must lie. The unique intersection of the two lines provided by the two sensors (provided that the two sensors and the object are not co-linear) determines the unique location of the object in space.

Consider now the situation in which we wish to determine the locations of p objects using two sensors. The first sensor would provide us with a set of lines L_1, $L_2, \ldots, L_p$ for the p objects and the second sensor would provide us a different set of lines $L'_1, L'_2, \ldots, L'_p$. To identify the location of the objects – using the fact that if two lines correspond to the same object, then the lines intersect one-another – we need to match the lines from the first sensor to the lines from the second sensor. In practice, two difficulties limit the use of this approach. First, a line from a sensor might intersect more than one line from the other sensor, so the matching is not unique. Second, two lines corresponding to the same object might not intersect because the sensors make measurement errors in determining the angle of sight. We can overcome this difficulty in most situations by formulating this problem as an assignment problem.

In the assignment problem, we wish to match the p lines from the first sensor with the p lines from the second sensor. We define the cost c_{ij} of the assignment (i, j) as the minimum Euclidean distance between the lines L_i and L_j. We can determine c_{ij} using standard calculations from geometry. If the lines L_i & L_j correspond to the same object, then c_{ij} would be close to zero. An optimal solution of the assignment problem would provide an excellent matching of the lines. Simulation studies have found that in most circumstances, the matching produced by the assignment problem defines the correct location of the objects.

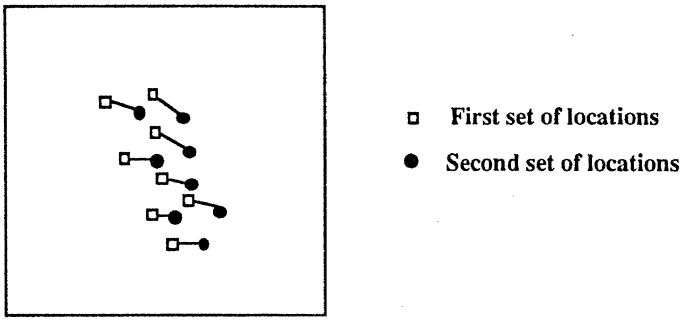


Fig. 21. Two snapshots of a set of 8 objects.

Application 17. Matching moving objects [Brogan, 1989; Kolitz, 1991]

In several different application contexts, we might wish to estimate the speeds and the directions of movement of a set of p objects (e.g., enemy fighter planes, missiles) that are moving in space. Using the method described in the preceding application, we can determine the location of the objects at any point in time. One plausible way to estimate the movement directions of the objects at which the objects are moving is to take two snapshots of the objects at two distinct times and then to match one set of points with the other set of points. If we correctly match the points, then we can assess the speed and direction of movement of the objects. As an example, consider Figure 21 which denotes the objects at time 1 by squares and the objects at time 2 by circles.

Let (x_i, y_i, z_i) denote the coordinates of object i at time 1 and (x_i', y_i', z_i') denote the coordinates of the same object at time 2. We could match one set of points with the other set of points in many ways. Minimizing the sum of the squared Euclidean distances between the matched points is reasonable in this scenario because it attaches a higher penalty to larger distances. If we take the snapshots of the objects at two times that are sufficiently close to each other, then the optimal assignment will often match the points correctly. In this application of the assignment problem, we let $N_1 = \{1, 2, \ldots, p\}$ denote the set of objects at time 1, let $N_2 = \{1', 2', \ldots, p'\}$ denote the set of objects at time 2, and we define the cost of an arc (i, j') as $[(x_i - x_i')^2 + (y_i - y_i')^2 + (z_i - z_i')^2]$. The optimal assignment in this graph will specify the desired matching of the points. From this matching, we obtain an estimate of the movement directions and speeds of the individual objects.

Application 18. Rewiring of typewriters [Machol, 1961]

For several years, a company had been using special electric typewriters to prepare punched paper tapes to enter data into a digital computer. Because the typewriter is used to punch a six-hole paper tape, it can prepare $2^6 = 64$ binary hole–no-hole patterns. The typewriters have 46 characters and each punches one of the 64 patterns. The company acquired a new digital computer that uses

different coding hole–no-hole patterns to represent characters. For example, using 1 to represent a hole and 0 to represent a no-hole, the letter A is 111100 in the code for the old computer and 011010 in the code for the new computer. The typewriter presently punches the former and must be modified to punch the latter.

Each key in the typewriter is connected to a steel code bar, and so changing the code of that key requires mechanical changes in the steel bar system. The extent of the changes depends upon how close the new and old characters are to each other. For the letter A, the second, third and sixth bits are identical in the old and new codes and no changes need be made for these bits; however, the first, fourth and fifth bits are different and so we would need to make three changes in the steel code bar connected to the A-key. Each change involves removing metal at one place and adding metal at another place. When a key is pressed, its steel code bar activates six cross-bars (which are used by all the keys) that are connected electrically to six hole punches. If we interchange the fourth and fifth wires of the cross-bars to the hole punches (which is essentially equivalent to interchanging the fourth and fifth bits of all characters in the old code), then we would reduce the number of mechanical changes needed for the A-key from three to one. However, this change of wires might increase the number of changes for some of the other 45 keys. The problem, then, is how to optimally connect the wires from the six cross-bars to the six punches so that we can minimize the number of mechanical changes on the steel code bars.

We formulate this problem as an assignment problem as follows. Define a network $G = (N_1 \cup N_2, A)$ with node sets $N_1 = \{1, 2, \ldots, 6\}$ and $N_2 = \{1', 2', \ldots, 6'\}$, and an arc set $A = N_1 x N_2$; the cost of the arc $(i, j') \in A$ is the number of keys (out of 46) for which the ith bit in the old code differs from the jth bit in the new code. Thus if we assign cross-bar i to the punch j, then the number of mechanical changes needed to correctly print the ith bit of each symbol is c_{ij}. Consequently, the minimum cost assignment will minimize the number of mechanical changes.

Additional applications

Additional applications of the assignment problem include: (1) *bipartite personnel assignment* [Machol, 1970; Ewashko & Dudding, 1971]; (2) *optimal depletion of inventory* [Derman & Klein, 1959]; (3) *scheduling of parallel machines* [Horn, 1973]; (4) *solving shortest path problems in directed networks* [Hoffman & Markowitz, 1963]; (5) *discrete location problems* [Francis & White, 1976]; and (6) *vehicle and crew scheduling* [Carraresi & Gallo, 1984].

7. Matchings

A *matching* in an undirected graph $G = (N, A)$ is a set of arcs with the property that every node is incident to at most one arc in the set; thus a matching induces

a pairing of (some of) the nodes in the graph using the arcs in A. In a matching, each node is matched with at most one other node, and some nodes might not be matched with any other node. In some applications, we want to match all the nodes (that is, each node must be matched to some other node); we refer to any such matching as a *perfect matching*. Suppose that each arc (i, j) in the network has an associated cost c_{ij}. The *(perfect) matching problem* seeks a matching that minimizes the total cost of the arcs in the (perfect) matching. Since any perfect matching problem can be formulated as a matching problem if we add the same large negative cost to each arc, in the following discussion, we refer only to matching problems.

Matching problems on bipartite graphs (i.e., on a graph $G = (N_1 \cup N_2, A)$, where $N = N_1 \cup N_2$ and each arc $(i, j) \in A$ has $i \in N_1$ and $j \in N_2$), are called *bipartite matching problems* and those on general graphs that need not be bipartite are called *nonbipartite matching problems*. There are two further ways of categorizing matching problems: *cardinality matching problems,* that maximize the number of pairs of nodes matched, and *weighted matching problems* that minimize the weight of the matching. The weighted matching problem on a bipartite graph is the same as the *assignment problem*, whose applications we described in the previous section. In this section, we describe applications of matching problems on nonbipartite graphs.

The matching problem arises in many different problem settings since we often wish to find the best way to pair objects or people together to achieve some desired goal. Some direct applications of nonbipartite matching problems include matching airplane pilots to planes, and assigning roommates to rooms in a hostel. We describe now three indirect applications of matching problems.

Application 19. Pairing stereo speakers [Mason & Philpott, 1988]

As a part of its manufacturing process, a manufacturer of stereo speakers must pair individual speakers before it can sell them as a set. The performance of the two speakers depends upon their frequency response. In order to measure the quality of the pairs, the company generates matching coefficients for each possible pair. It calculates these coefficients by summing the absolute differences between the responses of the two speakers at twenty discrete frequencies, thus giving a matching coefficient value between 0 and 30,000. Bad matches yield a large coefficient, and a good pairing produces a low coefficient.

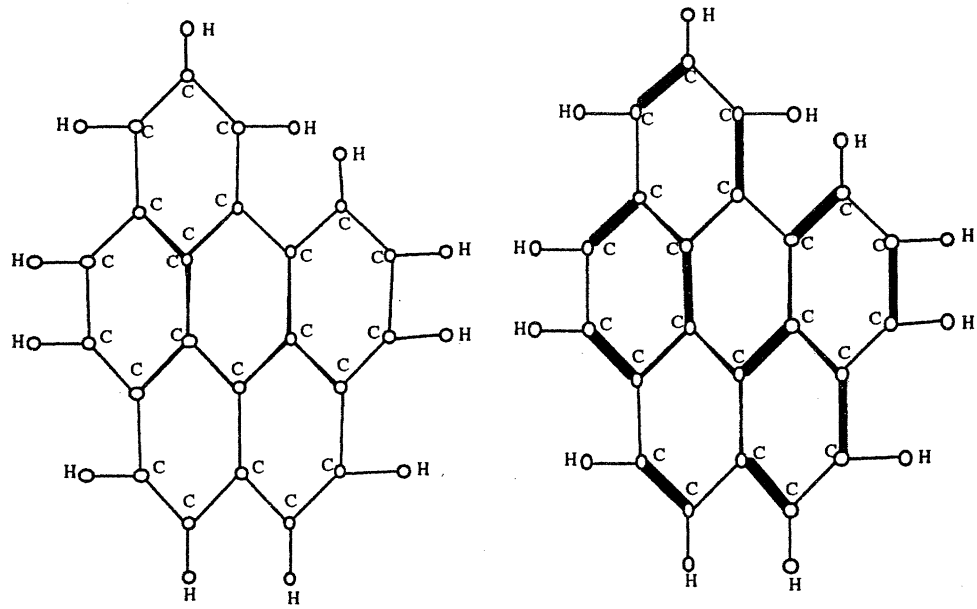
The manufacturer typically uses two different objectives in pairing the speakers: (i) finding as many pairs as possible whose matching coefficients do not exceed a specification limit; or (ii) pairing speakers within specification limits in order to minimize the total sum of the matching coefficients. The first objective minimizes the number of pairs outside of specification, and so the number of speakers that the firm must sell at a reduced price. This model is an application of the nonbipartite cardinality matching problem on an undirected graph: the nodes of this graph represent speakers and arcs join two nodes if the matching coefficients of the corresponding speakers are within the specification

limit. The second model is an application of the nonbipartite weighted matching problem.

Application 20. Determining chemical bonds [Dewar & Longuet-Higgins, 1952]

Matching problems arise in the field of chemistry as chemists attempt to determine the possible atomic structures of various molecules. Figure 22a specifies the partial chemical structure of a molecule of some hydrocarbon compound. The molecule contains carbon atoms (denoted by nodes with the letter 'C' next to them) and hydrogen atoms (denoted by nodes with the letter 'H' next to them). Arcs denote bonds between atoms. The bonds between the atoms, which can be either single or double bonds, must satisfy the 'valency requirements' of all the nodes. (The valency of an atom is the sum of its bonds.) Carbon atoms must have a valency of 4 and hydrogen atoms a valency of 1.

In the partial structure shown in Figure 22a, each arc depicts a single bond and, consequently, each hydrogen atom has a valency of 1, but each carbon atom has a valency of only 3. We would like to determine which pairs of carbon atoms to connect by a double bond so that each carbon atom has valency 4. We can formulate this problem of determining some feasible structure of double bonds to determining whether or not the network obtained by deleting the hydrogen atoms contains a maximum cardinality matching with all nodes are matched. Figure 22b gives one feasible bonding structure of the compound; the bold lines in this network denote double bonds between the atoms.



(a) (b)

Fig. 22. Determining the chemical structure of a hydrocarbon.

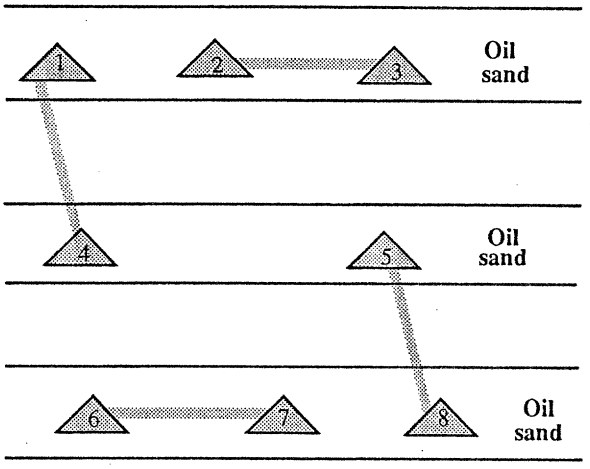


Fig. 23. Targets and matchings for the dual completion problem.

Application 21. Dual completion of oil wells [Devine, 1973]

An oil company has identified several individual oil traps, called *targets*, in an offshore oil field and wishes to drill wells to extract oil from these traps. Figure 23 illustrates a situation with eight targets. The company can extract any target separately (so called *single completion*) or extract oil from any two targets together by drilling a single hole (so called *dual completion*). It can estimate the cost of drilling and completing any target as a single completion or any pair of targets as a dual completion. This cost will depend on the three-dimensional spatial relationships of targets to the drilling platform and to each other. The decision problem is to determine which targets (if any) to drill as single completions and which pairs to drill together as duals, so as to minimize the total drilling and completion costs. If we restrict the solution to use only dual completions, then the decision problem is a nonbipartite weighted matching problem.

Application 22. Parallel savings heuristics [Ball, Bodin & Dial, 1983; Altinkemer & Gavish, 1991]

A number of combinatorial optimization problems have an underlying clustering structure. For example, in many facility location applications we wish to cluster customers into service regions, each served by a single facility (warehouses in distribution planning, concentrators in telecommunication systems design, hospitals within a city). In vehicle routing, we wish to form customer clusters, each representing the customer deliveries allocated to a particular vehicle. The cost for any particular cluster might be simple, for example, the total cost of traveling from each customer to its service center, or complex: in vehicle routing, the cost of a solution is the sum of the costs required to service each cluster (which usually is computed by solving a traveling salesman problem – see Section 12) plus a cost that depends on the number of clusters. Airline or mass transit crew scheduling provide

another example. In this context, we need to assign crews to flight legs or transit trips. A cluster is a set of flight legs or trips that forms the workload for one crew.

In these examples, as well as in other applications, the generic problem has an input set T together with a cost function $c(S)$ defined on subsets S of T. The problem is to partition T into subsets $S_1, S_2, \ldots, S_k$ in a way that minimizes the sum $c(S_1) + c(S_2) + \ldots + c(S_k)$.

In these settings, matching can be used as a tool for designing 'smart' heuristic solution procedures. A standard 'greedy' savings heuristic, for example, starts with an initial solution consisting of all subsets of size 1. Suppose we are given an intermediate solution that partitions the elements of T into disjoint subsets S_1, $S_2, \ldots, S_k$. For any pair of subsets, S_i and S_j, we define the savings obtained by combining S_i and S_j as:

$$c(S_i) + c(S_j) - c(S_i \cup S_j).$$

The general step of the greedy algorithm computes the savings obtained by combining all pairs of subsets and combines the two that provide the largest savings. A parallel savings algorithm considers the sets $S_1, S_2, \ldots, S_k$ simultaneously rather than just two at a time. At each iteration, it simultaneously combines the set of pairs – e.g., S_1 with S_6, S_2 with S_{11}, S_3 with S_4, etc. – that yields the greatest savings. We can find this set of combinations by solving a matching problem. The matching graph contains a node for each subset S_j. The graph contains the arc (i, j) whenever combining the two end nodes (i.e., the sets S_i and S_j) is feasible and yields positive savings. We allow nodes to remain unmatched. A maximum weight matching in this graph yields the set of combinations producing the greatest savings.

As stated, the savings algorithm always combines sets to form larger ones. Using a similar approach, we can construct a parallel savings improvement algorithm for this same class of problems. We start with any set of subsets that constitute a 'good' feasible solution (if we had obtained this partition by first using the parallel savings algorithm, combining no pair of subsets would induce a positive savings). We construct a matching graph as before, except that the savings associated with two subsets S_i and S_j becomes:

$$c(S_i) + c(S_j) - [c(S_i^*) + c(S_j^*)].$$

In this expression, S_i^* and S_j^* are the minimum cost partitions of $S_i \cup S_j$. We then replace S_i and S_j with S_i^* and S_j^*. We then iterate on this process until the matching graph contains no positive cost arcs.

In the most general setting, the minimum cost partitions could involve more than 2 sets. If finding the minimum cost partitions of $S_i \cup S_j$ is too expensive, we might instead use some heuristic method to find a 'good' partition of these sets (the parallel saving heuristic is a special case in which we always choose the partition as the single set $S_i \cup S_j$).

We then iterate on this process until the matching graph contains no positive cost edges.

Analysts have devised parallel savings algorithms and shown them to be effective for problems arising in vehicle routing, crew scheduling, and telecommunications.

Additional applications

Additional applications of the matching problems include: (1) *solving shortest path problems in undirected networks* [Edmonds, 1967]; (2) *the undirected Chinese postman problem* [Edmonds & Johnson, 1973]; (3) *two-processor scheduling* [Fujii, Kasami & Ninomiya, 1969]; (4) *vehicle and crew scheduling* [Carraresi & Gallo, 1984]; (5) *determining the rank of a matrix* [Anderson, 1975]; and (6) *making matrices optimally sparse* [Hoffman & McCormick, 1984].

8. Minimum spanning trees

As we noted previously, a spanning tree is a tree (i.e., a connected acyclic graph) that spans (touches) all the nodes of an undirected network. The cost of a spanning tree is the sum of the costs (or lengths) of its arcs. In the minimum spanning tree problem, we wish to identify a spanning tree of minimum cost (or length). Minimum spanning tree problems generally arise in one of two ways, directly or indirectly. In *direct* applications, we typically wish to connect a set of points using the least cost or least length collection of arcs. Frequently, the points represent physical entities that need to be connected to each other. In *indirect* applications, we either (i) wish to connect some set of points using a measure of performance that on the surface bears little resemblance to the minimum spanning tree objective (sum of arc costs), or (ii) the problem itself bears little resemblance to an 'optimal tree' problem – these instances often require creativity in modeling so that they become a minimum spanning tree problem. In this section, we consider several indirect applications. Section 12 on the network design problem describes several direct applications of the minimum spanning tree problem.

Application 23. Measuring homogeneity of bimetallic objects [Shier, 1982; Filliben, Kafadar & Shier, 1983]

This application shows how a minimum spanning tree problem can be used to determine the degree to which a bimetallic object is homogeneous in composition. To use this approach, we measure the composition of the bimetallic object at a set of sample points. We then construct a network with nodes corresponding to the sample points and with an arc connecting physically adjacent sample points. We assign a cost with each arc (i, j) equal to the product of the physical (Euclidean) distance between the sample points i and j and a homogeneity factor between 0 and 1. This homogeneity factor is 0 if the composition of the corresponding samples is exactly alike, and is 1 if the composition is very different; otherwise, it is a number between 0 and 1. Note that this measure gives greater weight to two points if they are different and are far apart. The cost of the minimum spanning tree is a measure of the homogeneity of the bimetallic object. The cost of the tree is 0 if all the sample points are exactly alike, and high cost values imply that the material is quite nonhomogeneous.

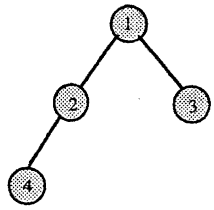


Fig. 24. Compact storage of a matrix.

Application 24. Reducing data storage (Kang, Lee, Chang & Chang, 1977]

In several different application contexts, we wish to store data specified in the form of a two-dimensional array more efficiently than storing all the elements of the array (that is, to save memory space). We assume that the rows of the array have many similar entries and differ only at a few places. Since the entities in the rows are similar, one approach for saving memory is to store one row, called the *reference row*, completely, and to store only the differences between some of the rows so that we can derive each row from these differences and the reference row. Let c_{ij} denote the number of different entries in rows i and j; that is, if we are given row i, then by making c_{ij} changes to the entries in this row we can obtain row j, and vice-versa. Suppose that the array contains four rows, represented by R_1, R_2, R_3 and R_4, and we decide to treat R_1 as a reference row. Then one plausible solution is to store the differences between R_1 and R_2, R_2 and R_4, and R_1 and R_3. Clearly, from this solution, we can obtain rows R_2 and R_3 by making c_{12} and c_{13} changes to the elements in row R_1. Having obtained row R_2, we can make c_{24} changes to the elements of this row to obtain R_4. We can depict this storage scheme in the form of a spanning tree shown in Figure 24.

It is easy to see that it is sufficient to store differences between those rows that correspond to arcs of a spanning tree. These differences permit us to obtain each row from the reference row. The total storage requirement for a particular storage scheme will be the length of the reference row (which we can take as the row with the least amount of data) plus the sum of the differences between the rows. Therefore, a minimum spanning tree would provide the least cost storage scheme.

Application 25. Cluster analysis (Gower & Ross, 1969; Zahn, 1971]

In this application, we describe the use of spanning tree problems to solve a class of problems that arises in the context of cluster analysis. The essential issue in cluster analysis is to partition a set of data into 'natural groups'; the data points within a particular group of data, or a cluster, should be more 'closely related' to each other than the data points not in that cluster. Cluster analysis is important in a variety of disciplines that rely upon empirical investigations. Consider, for example, an instance of a cluster analysis arising in medicine. Suppose we have data on a set of 350 patients, measured with respect to 18 symptoms. Suppose,

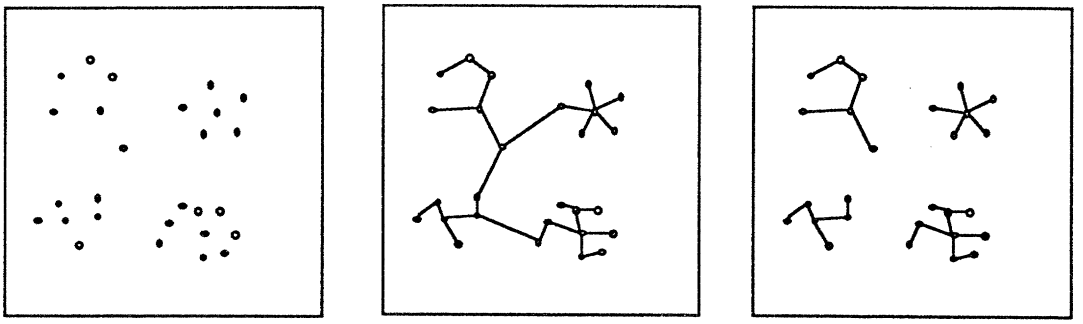


Fig. 25. Identifying clusters by finding a minimum spanning tree.

further, that a doctor has diagnosed all of these patients as having the same disease which is not well understood. The doctor would like to know if he or she can develop a better understanding of this disease by categorizing the symptoms into smaller groupings using cluster analysis. Doing so might permit the doctor to find more natural disease categories to replace or subdivide the original disease.

Suppose we are interested in finding a partition of a set of n points in two-dimensional Euclidean space into clusters. A popular method for solving this problem is by using Kruskal's algorithm for solving the minimum spanning tree problem [see, e.g., Ahuja, Magnanti & Orlin, 1993]. Kruskal's algorithm maintains a forest (i.e., a collection of node-disjoint trees) and adds arcs in a nondecreasing order of their costs. We can regard the components of the forest at intermediate steps as different clusters. These clusters are often excellent solutions for the clustering problem and, moreover, we can obtain them very efficiently. Kruskal's algorithm can be thought of providing n partitions: the first partition contains n clusters, each cluster containing a single point, and the last partition contains just one cluster containing all the points. Alternatively, we can obtain n partitions by starting with a minimum spanning tree and deleting tree arcs one by one in nonincreasing order of their lengths. We illustrate the later approach using an example. Consider a set of 27 points shown in Figure 25a. Suppose that Figure 25b shows a minimum spanning tree for these points. Deleting the three largest length arcs from the minimum spanning tree gives a partition with four clusters shown in Figure 25c.

Analysts can use the information obtained from the preceding analysis in several ways. The procedure we have described yields n partitions. Out of these, we might select the 'best' partition by simple visualization or by defining an appropriate objective function value. A good choice of the objective function depends upon the underlying features of the particular clustering application. We might note that this analysis is not limited to points in the two-dimensional space; we can easily extend it to multi-dimensional space if we define inter-point distances appropriately.

Application 26. System reliability bounds [Hunter, 1976; Worsley, 1982]

All systems/products are subject to failure. Typically, the reliability of a system (as measured by the probability that it will operate) depends upon the reliability of the system's individual components as well as the manner in which these components interact; that is, the reliability of the system depends upon the both the component reliabilities and the system structure. To model these situations, let us first consider a stylized problem setting with a very simple system structure. After analyzing this situation, we will comment on how this same analysis applies to more general, and often more realistic, systems.

In our simple setting, several components $k = 1, 2, \ldots, K$ of a system can perform the same function so a system fails only if all its components fail. Suppose we model this situation as follows. Let E_k denote the event that the kth component is operable and let E_k^c denote the complementary event that the kth component fails. Then, since the systems operates if and only if at least one of the components $1, 2, \ldots, K$ operates (or, equivalently, all its components don't fail),

$$\text{Prob(system operates)} = \text{Prob}\left(\bigcup_{k=1}^{K} E_k\right) = 1 - \text{Prob}\left(\bigcap_{k=1}^{K} E_k^c\right).$$

If component failures are independent events, then $\text{Prob}(\cap_{k=1}^{K} E_k^c) = \prod_{k=1}^{K} \text{Prob}(E_k^c)$, and so we can determine the system's operating probability if we know the failure probabilities of each component. In more complex situations, however, the component failure probabilities will be dependent (for example, because all the components wear together as we use the system or because the components are subject to similar environmental conditions (e.g., dust)).

In these situations, in theory we can compute the system operating probability using the principle of inclusion/exclusion (also known as the Boole or Poincaré formula), which requires knowledge of $\text{Prob}(\cap_{i \in S} E_i)$ for all subsets S of the components $k = 1, 2, \ldots, K$. In practice, the use of this formula is limited because of the difficulty of assessing the probability of joint events of the form $E_1 \cap E_2 \cap \ldots \cap E_q$, particularly for systems that have many components. As a result, analysts frequently attempt to find bounds on the system operating probability using less information than all the joint probabilities. One approach is to use more limited information, for example, the probability of single event of the form E_i and the joint probability of only two events at a time (i.e., events of the form $E_{ij} \equiv E_i \cap E_j$). Using this information, we have the following bound on the system's operating probability.

$$\text{Prob(system operates)} = \text{Prob}\left(\bigcup_{k=1}^{K} E_k\right) \leq \sum_{k=1}^{K} \text{Prob}(E_k).$$

Can we obtain better bounds using only the probabilities $\text{Prob}(E_{ij})$ of joint events?

Figure 26a shows a situation with three events. In this familiar Venn diagram, the three squares represent the three events, the intersection of two squares the joint events E_{ij}, and the intersection of all three squares the joint event $E_1 \cap E_2 \cap$

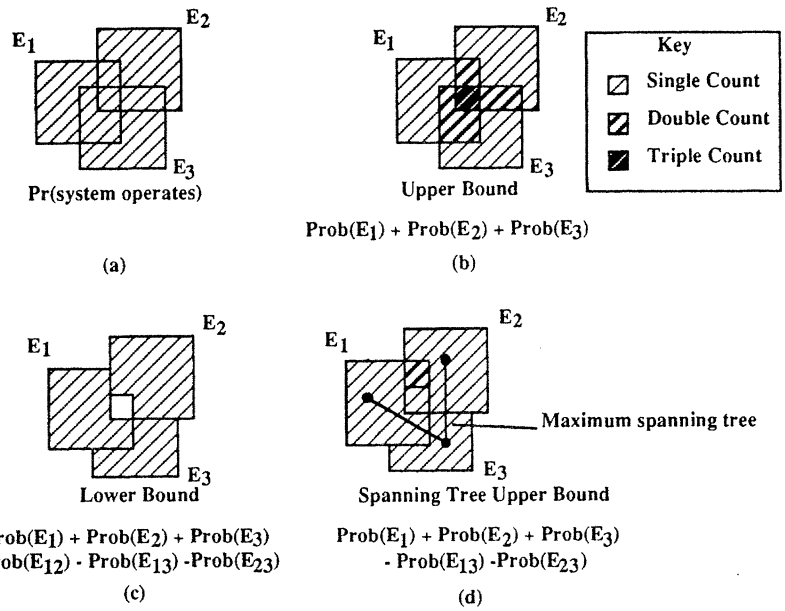


Fig. 26. Computing bounds on systems reliability.

E_3. Suppose, for convenience, that the area of any event (or joint event) equals the probability of that event. Summing the three events $\text{Prob}(E_1)$, $\text{Prob}(E_2)$, $\text{Prob}(E_3)$ gives an upper bound on the system's operating probability since the sum double counts the areas (probabilities) of the events $E_{ij} - E_1 \cap E_2 \cap E_3$ and triple counts the areas of the event $E_1 \cap E_2 \cap E_3$ (see the shading in Figure 26b). Note that the probability $\sum_{k=1}^{K} \text{Prob}(E_k) - \sum_{i<j} \text{Prob}(E_i \cap E_j)$ gives a lower bound on the system's operating probability since it by (generally) overcompensating for the double and triple counting: it eliminates the double counting of areas of all the events $E_{ij} - E_1 \cap E_2 \cap E_3$, but also eliminates all three counts of the event $E_1 \cap E_2 \cap E_3$ and so doesn't account for this event at all (see Figure 26c). Note, however, that in this case if instead of subtracting all three events of the form $\text{Prob}(E_{ij})$ from $\sum_{k=1}^{K} \text{Prob}(E_k)$, we subtract any two of them, then as shown by the shading in Figure 26d, we obtain an upper bound on the system operating probability.

We can generalize this observation for situations with more than three events E_k as follows: suppose we construct an undirected network with a node j for each event E_j. For any two events E_i and E_j, the network contains an arc (i, j) with a cost $c_{ij} = \text{Prob}(E_{ij})$. If T is any spanning tree on this network, we then obtain the *spanning tree upper bound*

$$\text{Prob}\left(\bigcup_{k=1}^{K} E_k\right) \leq \sum_{k=1}^{K} \text{Prob}(E_k) - \sum_{(i,j)\in T} \text{Prob}(E_{ij}).$$

To see that this bound is valid, let us decompose the events $E_1, E_2, \ldots, E_K$ into disjoint events $F_1, F_2, \ldots, F_Q$ satisfying the following property: no subset F_i is partially contained within a subset E_j for any i and j, that is either $F_i \subseteq E_j$ or else

$F_i \cap E_j = \emptyset$. For example, in Figure 26, the subset $E_1 \cap E_2$ is partially contained within E_3, but it can be decomposed into two disjoint subsets $F_1 = E_1 \cap E_2 \cap E_3$ and $F_2 = E_1 \cap E_2 - E_3$. Neither F_1 nor F_2 is partially contained within a subset E_j. In general, it is easy to decompose the events $E_1, E_2, \ldots, E_K$ in this manner.

Since $F_i \cap F_j = \emptyset$ whenever $i \neq j$,

$$\text{Prob(system operates)} = \\ = \text{Prob}\left(\bigcup_{k=1}^{K} E_k\right) = \text{Prob}\left(\bigcup_{q=1}^{Q} F_q\right) = \sum_{q=1}^{Q} \text{Prob}(F_q),$$

and so we wish to show that

$$\sum_{q=1}^{Q} \text{Prob}(F_q) \leq \sum_{k=1}^{K} \text{Prob}(E_k) - \sum_{(i,j)\in T} \text{Prob}(E_{ij}).$$

To see that this inequality is valid, note that each region F_i contributes $\text{Prob}(F_i)$ to the left-hand side of the inequalities. Now let us consider what it contributes to the right-hand side. Without loss of generality, renumber the events so that F_i is contained in events $E_1, \ldots, E_r$ for some $r \geq 1$. Then the region F_i contributes *exactly* $r\text{Prob}(F_i)$ to the term $\sum_{k=1}^{K} \text{Prob}(E_k)$. F_i also contributes $\text{Prob}(F_i)$ to each term $\text{Prob}(E_{jk})$ on the right-hand side when $(j, k) \in T$ with $1 \leq j, k \leq r$ since in these circumstances $F_i \subseteq E_{jk}$. However, because the set of arcs $(j, k) \in A$ with $1 \leq j, k \leq r$ forms a subgraph with only r nodes, it contains at most $r - 1$ arcs (j, k) from the tree T. We conclude that F_i contributes at most $(r - 1)\text{Prob}(F_i)$ to $\sum_{(i,j)\in T} \text{Prob}(E_{ij})$, and so the contribution of F_i to the right-hand side of the inequality is at least $\text{Prob}(F_i)$.

To obtain the best spanning tree upper bound, we choose T as the maximum spanning tree with respect to the costs c_{ij}. Figure 26d shows the network for the three event example. In this case, the figure provides the upper bound for the maximum spanning tree with the arcs (1,3) and (2,3). We would obtain inferior, but valid upper bounds by choosing any other spanning tree, for example, the tree with the arcs (1,2) and (2,3). Using this approach, by assessing the joint probabilities of two events at a time, and by performing a maximum spanning tree computation, we obtain an improved upper bound on the system's operating probability.

In many contexts, the system structure is more complicated than the simple structure we have considered in which the system operates only if one of its components operates. 'Coherent binary systems' provide one such generic set of applications. One way to represent the system structure in this setting is to produce a list of 'pathsets': a pathset is a component subset S satisfying the property that if all elements in S operate then so does the system. (In the simple setting we have analyzed, each set S is a single component.) If we assume that all component states are independent, then the probability that an individual pathset operates is the product of the reliabilities of the components contained in that pathset. In more general problem settings, the components in any pathset might not be independent and so obtaining the probability that any pathset operates

might be more difficult. In any event, if E_k represents the event that the kth pathset operates, then we can use our prior analysis to obtain the system reliability from the pathset reliabilities. Further development of this approach (particularly, how to define pathsets) would take us beyond the scope of our coverage here. We simple note that the analysis we have considered applies to much more general problems settings, including problem contexts broader than system reliability.

Additional applications

Some additional applications of the minimum spanning tree problem arise in the following situations: (1) *optimal message passing* [Prim, 1957]; (2) *all-pairs minimax path problem* [Hu, 1961]; (3) *solving a special case of the traveling salesman problem* [Gilmore & Gomory, 1964]; (4) *chemical physics* [Stillinger, 1967]; (5) *Lagrangian relaxation techniques* [Held & Karp, 1970]; (6) *network reliability analysis* [Van Slyke & Frank, 1972]; (7) *pattern classification* [Dude & Hart, 1973]; (8) *picture processing* [Osteen & Lin, 1974]; and (9) *network design* [Magnanti & Wong, 1984]. The survey paper of Graham & Hell [1985] provides references for additional applications of the minimum spanning tree problem.

9. Convex cost flows

In the minimum cost flow problem, we assume that the cost of the flow on any arc varies linearly with the amount of flow. Convex cost flow problems have a more general cost structure: the cost is a convex function of the flow. Flow costs vary in a convex manner in numerous problem settings including (i) power losses in a electrical network due to resistance; (ii) congestion costs in a city transportation network; and (iii) expansion costs of a communication network.

Many of the linear network flow models that we have considered in our previous discussions arise in more general forms with nonlinear costs. System congestion and queueing effects are one source of these nonlinearities (since queueing delays vary nonlinear with flows). In finance, we often are interested in not only the returns on various investments, but also in their risks, which analysts often measure by quadratic functions. In some other applications, cost functions assume different forms over different operating ranges, and so the resulting cost function is piecewise linear. For example, in production applications, the cost of satisfying customers demand is different if we meet the demand from current inventory or by backordering items. To give a flavor of the applications of convex cost network flow models, in this section, we describe three applications.

Application 27. Urban traffic flows (Magnanti, 1984; Sheffi, 1985; Florian & Hearn, 1992]

In road networks, as more vehicles use any road segment, the road become increasingly congested and so the delay on that road increases. For example, the

delay on a particular road segment, as a function of the flow x on that road, might be $ax/(u - x)$. In this expression, u denotes a theoretical capacity of the road and a is another constant: as the flow increases, so does the delay; moreover, as the flow x approaches the theoretical capacity of that road segment, the delay on the link becomes arbitrarily large. In many instances, as in this example, the delay function on each road segment is a convex function of the road segment's flow and so finding the flow plan that achieves the minimum overall delay, summed over all road segments, is a convex cost network flow model.

The most general model of this form has a multicommodity flow structure (see Section 11) with a commodity, and hence a separate set of mass balance equations (1b), defined for each pair of nodes that serve as each origins and destinations for travelers in the transportation system. In this case, the congestion effects (e.g., the function of the form $ax/(u - x)$) applies to the total flow x on any arc, which is the sum of flows by all commodities. The problem is a single commodity model whenever all the trips either originate or terminate at a single node; the flow x on any arc then is the total number of trips from the common origin (or destination) that use that arc.

Another model of urban traffic flow rests upon the behavioral assumption that users of the system will travel, with respect to any prevailing system flow, from their origin to their destination using a path with the minimum delay. So if $C_{ij}(x_{ij})$ denotes the delay on arc (i, j) as a function of the arc's flow x_{ij}, each user of the system will travel along a shortest delay path with respect to the total delay cost $C_{ij}(x_{ij})$ on the arcs of that path. This problem is a complex equilibrium model because the delay that one user incurs depends upon the flow of other users, and all of the users are simultaneously choosing their shortest paths. In this problem setting, we can find the equilibrium flow by solving a convex network flow model with the objective function

$$\sum_{(i,j)\in A} \int_0^{x_{ij}} C_{ij}(y)\,\mathrm{d}y.$$

If the delay function is nondecreasing for each variable x_{ij}, then the integral within the summation is convex and, since the sum of convex functions is convex, the overall objective function is convex. Moreover, if we solve the network optimization problem defined by this objective function and the network flow constraints, the optimality conditions are exactly the shortest path conditions for the users. (See the references for the details of these claims.)

This example is a special case of a more general result, known as *variational principle*, that arises in many settings in the physical and social sciences. The variational principle says that to find an equilibrium of a system, we can solve an associated optimization problem: the optimality conditions for the problem are equivalent with the equilibrium conditions.

Application 28. Matrix balancing [Schneider & Zenios, 1990]

Statisticians, social scientists, agricultural specialists and many other practitioners frequently use experimental data to try to draw inferences about the effects of certain control parameters at their disposal (for example, the effects of using different types of fertilizers). These practitioner often use contingency tables to classify items according to several criteria, with each criteria partitioned into a finite number of categories. As an example, consider classifying r individuals in a population according to the criteria of marital status and age, assuming that we have divided these two criteria into p and q categories, respectively. As categories for marital status, we might choose single, married, separated, or divorced; and as categories for age, we might choose below 20, 21 – 25, 26 – 30, and so forth. This categorization gives a table of pq cells. Dividing the entry in each cell by the total number of items r in the population gives the probability

In many applications, it is important to estimate the current cell probabilities which continually change with time. We can estimate these cell probabilities very accurately using a census, or approximately by using a statistical sampling of the population; however, even this sampling procedure is expensive. Typically, we would calculate the cell probabilities by sampling only occasionally (for some applications, only once in several years), and at other times revise the most recent cell probabilities based on partial observations. Suppose we let a_{ij} denote the most recent cell probabilities. Suppose, further, that we know some aggregate data in each category with high precision; in particular, suppose we know the row sums and column sums. Let u_i denote the number of individuals in the ith marital category and let v_j denote the number of individuals in the jth age category. Let $r = \sum_{i=1}^{p} u_i$. We want to obtain an estimate x_{ij}'s of the current cell probabilities so that the cumulative sum of the cell probabilities for ith row equals u_i/r, the cumulative sum of the cell probabilities for jth column equals v_j/r, and the matrix x is, in a certain sense, *nearest* to the most recent cell probability matrix α. One popular measure of defining the *nearness* is to minimize the weighted cumulative squared deviation of the individual cell probabilities. With this objective, our problem reduces to the following convex cost flow problem:

$$\text{minimize} \quad \sum_{i=1}^{p} \sum_{j=1}^{q} w_{ij}(x_{ij} - \alpha_{ij})^2$$

subject to

$$\sum_{j=1}^{q} x_{ij} = \frac{u_i}{r}, \quad \text{for all } i = 1, \ldots, p,$$

$$\sum_{i=1}^{p} x_{ij} = -\frac{v_j}{r}, \quad \text{for all } j = 1, \ldots, q,$$

$$x_{ij} \geq 0, \quad \text{for all } i = 1, \ldots, p, \text{ and for all } j = 1, \ldots, q.$$

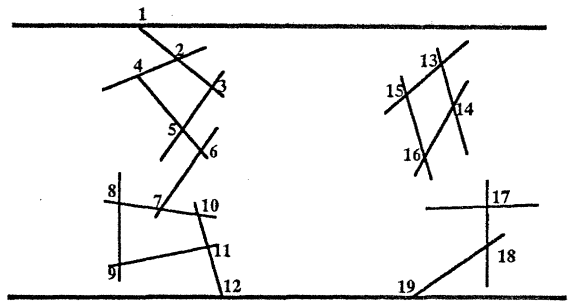
This type of *matrix balancing problem* arises in many other application settings. The inter-regional migration of people provides another important application. In the United States, using the general census of the population, taken once every ten years, the federal government produces flow matrices with detailed migration characteristics. It uses this data for a wide variety of purposes, including the allocation of federal funds to the states. Between the ten year census, net migration estimates for every region become available as by-products of annual population estimates. Using this information, the federal government updates the migration matrix so that it can reconcile the out-of-date detailed migration patterns with the more recent net figures.

Application 29. Stick percolation problem [Ahlfeld, Dembo, Mulvey & Zenios, 1987]

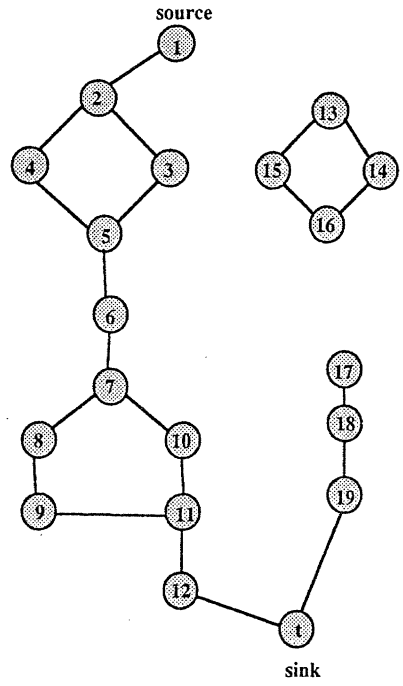
One method for improving the structural properties of (electrically) insulating materials is to embed sticks of high strength in the material. The current approach used in practice is to add, at random, sticks of uniform length to the insulating material. Because the sticks are generally conductive, if they form a connected path from one side of the material to the other, then the configuration will destroy the material's desired insulating properties. Material scientists call this phenomenon *percolation*. Although longer sticks offer better structural properties, they are more likely to cause percolation. Using simulation, analysts would like to know the effect of stick length on the possibility that a set of sticks causes percolation, and when percolation does occur what is the resulting heat loss due to (electrical) conduction.

Analysts use simulation in the following manner: the computer randomly places p sticks of a given length L in a square region; see Figure 27a for an example. We experience heat loss because of the flow of current through the intersection of two sticks. We assume that each such intersection has unit resistance. We can identify whether percolation occurs and determine the associated power dissipation by creating an equivalent resistive network as follows. We assign a resistor of one unit to every intersection of the sticks. We also associate a current source of one unit with one of the boundaries of the insulant and a unit sink with the opposite boundary. The problem then becomes of determining the minimum power dissipation of the resistive network.

Figure 27b depicts the transformation of the stick percolation problem into a network model. In this network model, each node represents a resistance and contributes to the power dissipation. The node splitting transformation described in Section 2 permits us to model the node resistances as arc resistances. Recall from Ohm's law that a current flow of x amperes across a resistor of r Ω creates a power dissipation of rx^2 W. Moreover, the current flows in an electrical network in a way that minimizes the rate of power dissipation (i.e., follows the path of least resistance). Consequently, we can state the stick percolator problem as the following convex cost flow problem.



(a)



(b)

Fig. 27. Formulating the stick percolator problem. (a) Placement of sticks. (b) Corresponding network.

$$\text{minimize} \quad \sum_{(i,j)\in A} r_{ij} x_{ij}^k$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1, & \text{for } i = s \\ 0, & \text{for all } i \in N - \{s \text{ and } t\} \\ -1, & \text{for } i = t \end{cases}$$

$$x_{ij} \geq 0, \quad \text{for all } (i, j) \in A.$$

In this model, x_{ij} is the current flow on arc (i, j). The solution of this convex cost flow model indicates whether percolation occurs (that is, if the problem has a feasible solution), and if so, then the solution specifies the value of the associated power loss.

Additional applications

Some additional applications of the convex cost flow problem are (1) *flows in electrical networks* [Hu, 1966]; (2) *area transfers in communication networks* [Monma & Segal, 1982]; (3) *the target-assignment problem* [Manne, 1958]; (4) *solution of Laplace's equation* [Hu, 1967]; (5) *production scheduling problems* [Ratliff, 1978]; (6) *pipeline network analysis problem* [Collins, Cooper, Helgason, Kennington & Leblanc, 1978]; (7) *microdata file merging* [Barr & Turner, 1981]; and (8) *market equilibrium problems* [Barros & Weintraub, 1986]. Papers by Ali, Helgason & Kennington [1978], Dembo, Mulvey & Zenios [1989], and Schneider & Zenios [1990] provide additional references on applications of the convex cost flow problem.

10. Generalized flows

In the minimum cost flow problem, arcs conserve flows, i.e., the flow entering an arc equals the flow leaving the arc. In generalized flow problems, arcs might 'consume' or 'generate' flow. If x_{ij} units of flow enter an arc (i, j), then $m_{ij}x_{ij}$ units arrive at node j; m_{ij} is a positive *multiplier* associated with the arc. If $0 < m_{ij} < 1$, then the arc is *lossy*, and if $1 < m_{ij} < \infty$, then the arc is *gainy*. Generalized network flow problems arise in several application contexts: for example (i) power transmission through electric lines where power is lost with distance; (ii) flow of water though pipelines or canals that lose water due to seepage or evaporation; (iii) transportation of a perishable commodity; and (iv) cash management scenarios in which arcs represent investment opportunities and multipliers represent appreciation or depreciation of an investment's value.

Generalized networks can successfully model many application settings that cannot adequately be represented as minimum cost flow problems. Two common interpretations of the arc multipliers underlie many uses of generalized flows. In the first interpretation, we view the arc multipliers as modifying the amount of flow of some particular item. Using this interpretation, generalized networks model situations involving physical transformations such as evaporation, seepage, deterioration, and purification processes with various efficiencies, as well as administrative transformations such as monetary growth due to interest rates. In the second interpretation, we view the multiplication process as transforming one type of item into another. This interpretation allows us to model processes such as manufacturing, currency exchanges, and the translation of human resources into job requirements. The following applications of generalized network flows use one or both of these interpretations of the arc multipliers.

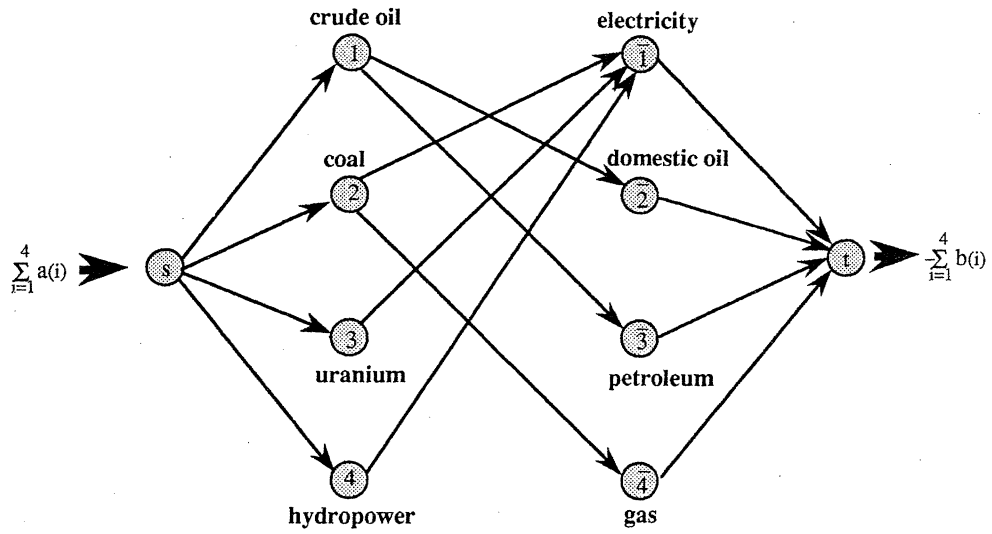


Fig. 28. Energy problem as a generalized network flow problem.

Application 30. Determining an optimal energy policy [Gondran & Minoux, 1984]

As part of their national planning effort, most countries need to decide upon an energy policy, i.e., how to utilize the available raw materials to satisfy their energy needs. Assume, for simplicity, that a particular country has four basic raw materials: crude oil, coal, uranium, and hydropower; and it has four basic energy needs: electricity, domestic oil, petroleum, and gas. The country has the technological base and infrastructure to convert each raw material into one or more energy forms. For example, it can convert crude oil into domestic oil or petrol, coal into electricity, and so forth. The available technology base specifies the efficiency and the cost of each conversion. The objective is to satisfy, at the least possible cost of energy conversion, certain annual consumption levels of various energy needs from a given annual production of raw materials.

Figure 28 shows the formulation of this problem as a generalized network flow problem. The network has three types of arcs: (i) *source arcs* (s, i) emanating from the source node s; (ii) *sink arcs* $(\bar{j}, t)$ entering the sink node t; and *conversion arcs* $(i, \bar{j})$. The source arc (s, i) has a capacity equal to the availability $a(i)$ of the raw material i and a flow multiplier of value one. The sink arc $(\bar{j}, t)$ has capacity equal to the demand $b(j)$ of type j energy need and flow multiplier of value one. Each conversion arc $(i, \bar{j})$ represents the conversion of raw material i into the energy form j; the multiplier of this arc is the efficiency of the conversion, i.e., units of energy j obtained from one unit of raw material i; the cost of the arc $(i, \bar{j})$ is the cost of this conversion.

Application 31. Machine loading [Dantzig, 1962]

Machine loading problems arise in a variety of application domains. In one of the most popular contexts, we would like to schedule the production of r products

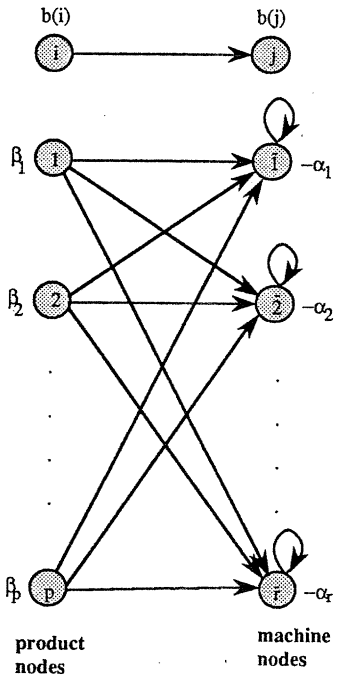


Fig. 29. Formulating a machine loading problem as a generalized network flow problem.

on p machines. Suppose that machine i is available for a_i hours and that any of the p machines can produce each product. Producing one unit of product i on machine j consumes a_{ij} hours of the machine's time and costs c_{ij} dollars. To meet the demands of the products, we must produce β_j units of product j. In the machine loading problem, we wish to determine how we should produce, at the least possible production cost, r products on p machines.

In this problem setting, products compete with each other for the use of the more efficient, faster machines; the limited availability of these machines forces us to use the less economical and slower machines to process some of the products. To achieve the optimal allocation of products to the machines, we can formulate the problem as a generalized network flow problem as shown by Figure 29. The network has p product nodes, 1, 2, ..., p, and r machine nodes, $\overline{1}$, $\overline{2}$, ..., $\overline{r}$. Product node i is connected to every machine node $\overline{j}$. The multiplier a_{ij} on arc $(i, \overline{j})$ indicates the hours of machine capacity needed to produce one unit of product i on machine j. The cost of the arc $(i, \overline{j})$ is c_{ij}. The network also has arcs $(\overline{j}, \overline{j})$ for each machine node $\overline{j}$; the multiplier of each of these arcs in 2 and the cost is zero. The purpose of these arcs is to account for the unfulfilled capacities of the machines: we can send additional flow along these arcs to generate enough flow (or items) to exactly consume the available machine capacity.

Application 32. Managing warehousing goods and funds flows [Cahn, 1948]

An entrepreneur owns a warehouse of fixed capacity H that she uses to store a price-volatile product. Knowing the price of this product over the next K time

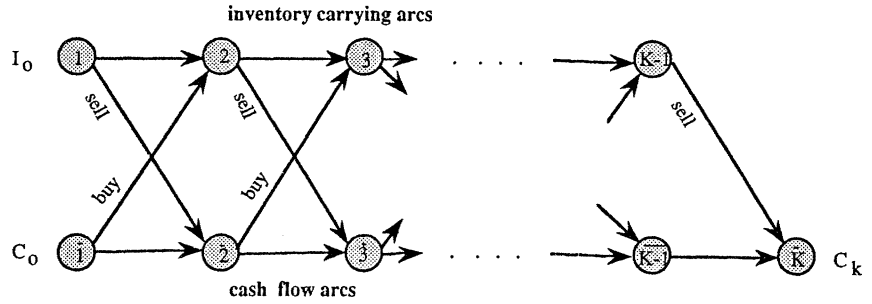


Fig. 30. Formulating the warehouse funds flow model.

periods, she needs to manage her purchases, sales, and storage patterns. Suppose that she holds I_0 units of the good and C_0 dollars as her initial assets. In each period, she can either buy more goods, or sell the goods in the warehouse to generate additional cash. The price of the product varies from period to period and ultimately all goods must be sold. The problem is to identify a buy–sell strategy that maximizes the amount of cash C_K available at the end of the Kth period.

Figure 30 gives a generalized network flow formulation of the warehousing goods and funds flow problem. This formulation has the following three types of arcs:

(i) *Inventory carrying arcs.* The cost of this type of arc is the inventory carrying cost per unit; its capacity is H. The multiplier on this arc is less than one or equal to one, depending upon whether carrying inventory incurs any loss.

(ii) *Cash flow arcs.* These arcs are uncapacitated and have zero cost. The multiplier of this type of arc is the bank interest rate for that period.

(iii) *Buy and sell arcs.* These arcs also are uncapacitated and have zero cost. If p_i is the purchase cost of the product in period i, then a buy arc has a multiplier of value $1/p_i$ and a sell arc has a multiplier of value p_i.

It is easy to establish a one-to-one correspondence between buy–sell strategies of the entrepreneur and flows in the underlying network. To enrich this model, we could easily incorporate various additional features such as policy limits on the maximum and minimum amount of cash held or goods flow in each period or introduce alternative investments such as certificates of deposits that extend beyond a single period.

Additional applications

Additional applications of the generalized network flow problem arise in (1) *resort development* [Glover & Rogozinski, 1982]; (2) *airline seat allocation problems* [Glover, Hultz, Klingman & Stutz, 1978; Dror, Trudeau & Ladany, 1988]; (3) *personnel planning* [Gorham, 1963]; (4) *a consensus ranking model* [Barzilai, Cook & Kress, 1986]; (5) *cash flow management in an insurance company* [Crum & Nye, 1981]; and (6) *land management* [Glover, Glover & Martinson, 1984]. The survey papers of Glover, Hultz, Klingman & Stutz [1978], and Glover,

Klingman & Phillips [1990] contain additional references concerning applications of generalized network flow problems.

11. Multicommodity flows

The minimum cost flow problem models the flow of a single commodity over a network. Multicommodity flow problems arise when several commodities use the same underlying network. The commodities might be differentiated either by their physical characteristics or simply by their origin–destination pairs. Different commodities have different origins and destinations, and commodities have separate mass balance constraints at each node. However, the sharing of common arc capacities binds the commodities together. In fact, the essential issue addressed by the multicommodity flow problem is the allocation of the capacity of each arc to the individual commodities in a way that minimizes overall flow costs. Multicommodity flow problems arise in many practical situations including (i) the transportation of passengers from different origins to different destinations within a city; (ii) the routing of nonhomogeneous tankers (nonhomogeneous in terms of speed, carrying capability and operating costs); (iii) the worldwide shipment of different varieties of grains (such as corn, wheat, rice, and soybeans) from countries that produce grain to those that consume it; and (iv) the transmission of messages in a communication network between different origin–destination pairs.

Multicommodity flow problems arise in a wide variety of application contexts. In this section, we consider several instances of one very general type of application as well as a racial balancing example and a vehicle fleet planning example.

Application 33. Routing of multiple commodities [Golden, 1975; Crainic, Ferland & Rousseau, 1984]

In many applications of the multicommodity flow problem, we distinguish commodities because they are different physical goods, and/or because they have different points of origin and destination; that is, either (i) several physically distinct commodities, for example different manufactured goods, share a common network, or (ii) a single physical good (e.g., messages or products) flows on a network, but the good has multiple points of origin and destination defined by different pairs of nodes in the network that need to send the good to each other. This second type of application arises frequently in problem contexts such as communication systems or distribution/transportation systems. In this section, we briefly introduce several application domains of both types.

In some of these applications, the costs sometimes are convex, as in the case of the urban traffic equilibrium (see Application 27), due to congestion effects (e.g., delays) on arc flows.

Communication networks. In a communication network, nodes represent origin and destination stations for messages, and arcs represent transmission lines.

Messages between different pairs of nodes define distinct commodities; the supply and demand for each commodity is the number of messages to be sent between the origin and destination nodes of that commodity. Each transmission line has a fixed capacity (in some applications the capacity of each arc is fixed, in others, we might be able to increase the capacity at a certain cost per unit). In this network, the problem of determining the minimum cost routing of messages is a multicommodity flow problem.

Computer networks. In a computer communication network, the nodes represent storage devices, terminals, or computer systems. The supplies and demands correspond to the data transmission rates between the computer, terminals, and storage devices, and the transmission line capacities define the bundle constraints.

Railroad transportation networks. In a rail network, nodes represent yard and junction points, and arcs represent track sections between the yards. The demand is measured by the number of cars (or, any other equivalent measure of tonnage) to be loaded on any train. Since the system incurs different costs for different goods, we divide traffic demand into different classes. Each commodity in this network corresponds to a particular class of demand between a particular origin–destination pair. The bundle capacity of each arc is the number of cars that we can load on the trains that are scheduled to be dispatched on that arc (over some period of time). The decision problem in this network is to meet the demands of cars at the minimum possible operating cost.

Distribution networks. In distribution systems planning, we wish to distribute multiple (nonhomogeneous) products from plants to retailers using a fleet of trucks or railcars, and using a variety of railheads and warehouses. The products define the commodities of the multicommodity flow problem, and the joint capacities of the plants, warehouses, railyards, and the shipping lanes define the bundle constraints. Note that this application has important bundle constraints imposed upon the nodes (plants, warehouses) as well as the arcs.

Foodgrain export–import network. The nodes in this network correspond to geographically dispersed locations in different countries, and the arcs correspond to shipments by rail, truck, and ocean freighter. Between these locations, the commodities are various foodgrains such as corn, wheat, rice and soybean. The capacities at the ports define the bundle constraints.

Application 34. Racial balancing of schools [Clarke & Surkis, 1968]

In 1968, the Supreme Court of the United States ruled that all school systems in the country should begin admitting students to schools on a nondiscriminatory basis, and should employ faster techniques to promote desegregated schools across the nation. This decision made it necessary for many school systems to develop radically different procedures for assigning students to schools. Since the

Supreme Court did not specify what constitutes an acceptable racial balance, the individual school boards used their own best judgments to arrive at acceptable criteria upon which to base their desegregation plans. This application describes a multicommodity flow model for determining an optimal assignment of students to schools that minimizes the total distance travelled by the students, given a specification of lower and upper limits on the required racial balance in each school.

Suppose that a school district has S schools and the capacity of the jth school is u_j. For the purpose of this formulation, we divide the school district into L population centers. These locations might, for example, be census tracts, bus stops, or city blocks. The only restriction on the population centers is that they be finite in number, and that a single distance measure reasonably approximates the distance any student at center i must travel if he or she is assigned to school j. Let S_{ik} denote the available number of students of the kth ethnic group at the ith population center. The objective is to assign students to schools in a way that achieves the desired ethnic composition for each school and minimizes the total distance traveled by the students. Each school j has the ethnic requirement that it must enroll at least l_{jk} and no more than u_{jk} students from the kth ethnic group.

We can model this problem as a multicommodity flow problem on an appropriately defined network. Figure 31 shows this network representation for a problem with three population centers and three schools. This network has one node for each population center and for each school as well as a 'source' and a 'sink' node for each ethnic group. The flow commodities represent the students of different ethnic groups. The students of the kth ethnic group flow from the source a_k to the sink e_k via population center and school nodes. We set the upper bound on arc (a_k, b_i) connecting the kth ethnic group source node and the ith population center equal to s_{ik} and the cost of the arc (b_i, c_j) connecting the ith population

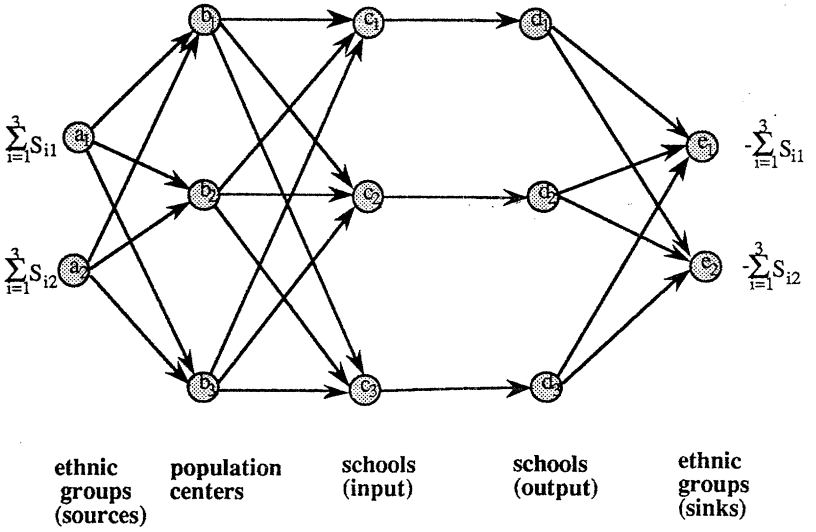


Fig. 31. Formulating the racial balancing problem as a multicommodity flow problem.

center and jth school equal to f_{ij}, the distance between that population center and that school. By setting the capacity of the arc (c_j, d_j) equal to u_j, we ensure that the total number of students (of all ethnic groups) allocated to the jth school do not exceed the maximum student population for this school. The students of all ethnic groups must share the capacity of each school. Finally, we incorporate constraints on the ethnic compositions of the schools by setting the lower and upper bounds on the arc (d_j, e_k) equal to l_{jk} and u_{jk}. It is fairly easy to verify that the multicommodity flow problem models the racial balancing problem and so a minimum multicommodity flow will specify an optimal assignment of students to the schools.

Application 35. Multivehicle tanker scheduling [Bellmore, Bennington & Lubore, 1971]

Suppose we wish to determine the optimal routing of fuel oil tankers required to achieve a prescribed schedule of deliveries: each delivery is a shipment of some commodity from a point of supply to a point of demand with a given delivery date. In the simplest form, this problem considers a single product (e.g., aviation gasoline or crude oil) to be delivered by a single type of tanker. As shown by Application 10, it is possible to determine the minimum tanker fleet to meet the delivery schedule for this simple version of the problem by solving a maximum flow problem. The multivehicle tanker scheduling problem, studied in this application, considers the scheduling and routing of a fixed fleet of nonhomogeneous tankers to meet a prespecified set of shipments of multiple products. The tankers differ in their speeds, carrying capabilities, and operating costs.

To formulate the multivehicle tanker scheduling problem as a multicommodity flow problem, we let the different commodities correspond to different tanker types. The network corresponding to the multivehicle tanker scheduling problem is similar to that of the single vehicle type, shown in Figure 32, except that each distinct type of tanker originates at a unique source node s^k. This network has four types of arcs (see Figure 32 for a partial example with two tankers types): in-service arcs, out-of-service arcs, delivery arcs, and return arcs. An in-service arc

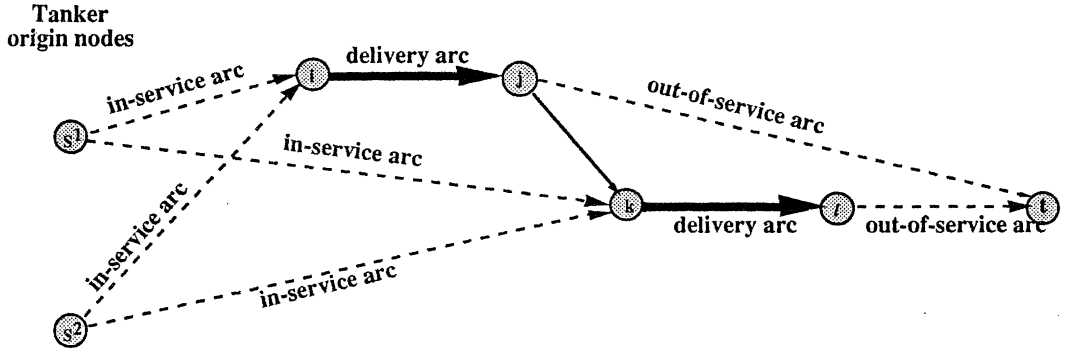


Fig. 32. Multivehicle tanker scheduling problem as a multicommodity flow problem.

corresponds to the initial employment of a tanker type; the cost of this arc is the cost of deploying the tanker at the origin of the shipment. Similarly, an out-of-service arc corresponds to the removal of the tanker from service. A delivery arc (i, j) represents a shipment from origin i to destination j; the cost c_{ij}^k of this arc is the operating cost of carrying the shipment by a tanker of type k. A return arc (j, k) denotes the movement ('backhaul') of an empty tanker, with an appropriate cost, between two consecutive shipments (i, j) and (k, l).

Each arc in the network has a capacity of one. The shipment arcs have a bundle capacity ensuring that at most one tanker type services that arc. Each shipment arc also has a lower flow bound of one unit, which ensures that the chosen schedule does indeed deliver the shipment. Some arcs might also have commodity-based capacities u_{ij}^k. For instance, if tanker type 2 is not capable of handling the shipment on arc (i, j), then we set $u_{ij}^2 = 0$. Moreover, if tanker type 2 can use the return arc (j, k), but the tanker type 1 cannot (because it is too slow to make the connection between shipments), then we set $u_{jk}^1 = 0$.

Airline scheduling is another important application domain for this type of model. In this problem context, the vehicles are different type of airplanes in an airline's fleet (for example, Boeing 727s or 747s or McDonald Douglas DC 10s). The delivery arcs in this problem context are the flight legs that the airline wishes to cover.

We might note that in this formulation of the multivehicle tanker scheduling problem, we are interested in integer solutions of the multicommodity flow problem. The solutions obtained by the multicommodity flow algorithms [see Kennington & Helgason, 1980; Ahuja, Magnanti & Orlin, 1993] need not be integral. Nevertheless, the fractional solution might be useful in several ways. For example, we might be able to convert the non-integral solution into a (possibly suboptimal) integral solution by minor tinkering or we might use the non-integral solution as a bound in solving the integer-valued problem by a branch and bound enumeration procedure.

Additional applications

Additional applications of the multicommodity flow problem include: (1) *warehousing of seasonal products* [Jewell, 1957]; (2) *optimal deployment of resources* [Kaplan, 1973]; (3) *multiproduct multistage production-inventory planning* [Evans, 1977]; (4) *multicommodity distribution system design* [Geoffrion & Graves, 1974]; (5) *rail freight planning* [Bodin, Golden, Assad & Ball, 1983; and Assad, 1980]; and (6) *VLSI chip design* [Korte, 1988].

12. The traveling salesman problem

The traveling salesman problem (TSP) is perhaps the most well known problem in the field of network optimization, or in fact in all of operations research. The problem is deceptively easy to state: We are given n cities denoted 1, 2, 3, ..., n, as

well as an inter city distance c_{ij} between each pair i, j of cities. The problem is to determine a tour (cycle) that passes through each city exactly once and minimizes the total distance traveled. In general this problem is known to be difficult to solve. In the parlance of computational complexity theory, it is NP-hard.

The simplicity of the problem belies the difficulty of solving it to optimality. The problem has attracted an immense amount of attention from researchers in combinatorial optimization, who have written hundreds of papers on this particular topic. In fact, the classic book on this problem, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, lists over 400 references.

There are a number of ways of mathematically describing the traveling salesman problems. We describe them in terms of permutations.

Let $N = \{1, 2, \ldots, n\}$. We represent a tour by a function t, letting $t(i)$ denotes the ith city visited on the tour. Let T denote the set of all tours. Using this notation, we can describe the traveling salesman problem as follows:

$$\text{minimize} \quad c_{t(n)t(1)} + \sum_{i=1}^{n-1} c_{t(i)t(i+1)}$$

$$\text{subject to} \quad t \in T.$$

In some problem instances, the tour need not return to its starting city. In this case, the tour is a path, often referred to as a *Hamiltonian path*, and we refer to the problem as the *minimum cost Hamiltonian path problem*, or more briefly as the *Hamiltonian path problem*. To formulate the Hamiltonian path problem mathematically, we can use of slight variant of the previous TSP formulation:

$$\text{minimize} \quad \sum_{i=1}^{n-1} c_{t(i)t(i+1)}$$

$$\text{subject to} \quad t \in T.$$

We might note that the Hamiltonian path problem is easily transformed into a traveling salesman problem. We simply add a dummy node whose distance from all other nodes is some constant, say 1.

Matrix representations of traveling salesman problems

To describe two of our applications, we use a particularly useful representation of the traveling salesman problem that we next describe.

Suppose that u and v are each vectors with m 0 or 1 components. The *hamming distance* from u to v, which we denote as $d(u, v)$, is the number of components in which u differs from v. For example, if $u = 1000110$ and $v = 1001100$, then $d(u, v) = 2$.

We say that a traveling salesman problem (or the Hamiltonian path problem) is matrix representable if each city i has an associated 0–1 vector A_i and $c_{ij} = d(A_i, A_j)$. Any traveling salesman problem with integral distances can be

matrix represented if we permit ourselves to add a constant M to each c_{ij} prior to representing it (we will not establish this fact). Suppose that a traveling salesman problem is matrix represented. Can we interpret the optimal tour in terms of the representing vectors? To illustrate how to do so, consider the following representing vectors.

$A_1 = 1\ 1\ 1\ 0$
$A_2 = 0\ 1\ 0\ 0$
$A_3 = 1\ 1\ 0\ 0$
$A_4 = 0\ 0\ 0\ 1$
$A_5 = 0\ 0\ 0\ 1$
$A_6 = 1\ 0\ 1\ 1$
$A_7 = 1\ 1\ 1\ 1$
$A_8 = 0\ 0\ 0\ 0$

In this case, $c_{12} = c_{21} = d(A_1, A_2) = 2$; $c_{13} = c_{31} = d(A_1, A_3) = 1$; and so forth.

In this context, the rows of the matrix correspond to the cities and a tour is a permutation of the rows. That is, $t(i)$ denote the ith row 'visited' by the tour. For example, we can represent the tour 1–2–3–7–6–5–4–8–1 by the following permuted version of the matrix:

$A_1 = 1\ 1\ 1\ 0$
$A_2 = 0\ 1\ 0\ 0$
$A_3 = 1\ 1\ 0\ 0$
$A_7 = 1\ 1\ 1\ 1$
$A_6 = 1\ 0\ 1\ 1$
$A_5 = 0\ 0\ 0\ 1$
$A_4 = 0\ 0\ 0\ 1$
$A_8 = 0\ 0\ 0\ 0$

This matrix has a row of all 0's, which we may assume without loss of generality to be the last city of the tour. Note that column 1 contributes a distance of 4 to the Hamiltonian path since its has different elements in rows 1 and 2, rows 2 and 3, rows 5 and 6, and rows components 8 and 1. In general, if the last row of the matrix consists of all 0's, each block of consecutive ones in a column contributes 2 to the tour length. Therefore, the traveling salesman distance is twice the number of consecutive blocks of ones in the columns.

Put another way, in this case the traveling salesman problem is equivalent to the problem of determining how to permute the rows of a matrix so that the columns of the resulting matrix has the fewest possible blocks of consecutive ones.

Application 36. Manufacturing of printed circuit boards [Magirou, 1986]

In the manufacturing of printed circuit boards (PCB's), lasers need to drill hundreds of small holes at pin locations. Subsequently, a manufacturer will solder electronic components to the PCB's at these points. To expedite the production of each chip, the laser must move efficiently from one drilling site to the next,

ultimately visiting each drilling site along its 'tour.' This manufacturing problem can be solved as a traveling salesman problem. More precisely, we can model the problem as a Hamiltonian path problem: the drill must visit each node at least once, but the first and last nodes need not be the same.

In practice, we would not list the between site transit times for each pair of drilling sites, but would instead list the coordinates of each drilling site. Typically, the distance between sites is the maximum of the distance from site i to site j in the x coordinate and the distance from site i to site j in the y coordinate. This distance models two independent motors, one working in the x direction and one working in the y direction.

This problem also arises in the manufacturing of VLSI circuits. The primary difference computationally is one of size. Bentley [1990] writes 'VLSI-designs in the near future could employ the solution of 10^7-city TSPs.'

Application 37. Identifying time periods for archeological finds [Shuchat, 1984]

One part of an archeologist's work is to assign dates to archeological finds or deposits. Consider a collection of deposits from m different excavation sites containing artifacts (e.g., pottery, jewelry), sorted into $n-1$ different types. We assume that each artifact type is associated with an (unknown) time period during which it was in use, and that each excavation site contains artifacts from consecutive (or nearly consecutive) time periods.

Associated with type i is a vector A_i, whose jth component a_{ij} is one if excavation site j contains an artifact of type i and is zero otherwise. If the artifact types were in their true order, then the ones of each column of the corresponding matrix would be in consecutive (or nearly consecutive) time positions. As in the matrix-represented TSP discussion, we introduce an artificial type n with an associated vector A_n of all 0's. Without loss of generality, we assume that n will be the last node of the tour. If t denotes a traveling salesman tour, the interpretation is that the oldest artifact is $t(1)$, the second oldest is $t(2)$, and so forth. A gap between two blocks of consecutive ones corresponds to a time period omitted from an excavation site between two time periods that are represented at the site.

By solving the traveling salesman problem, we are assigning periods to types and thus implicitly permuting the rows of A so as to minimize the number of blocks of consecutive ones in the resulting matrix. Equivalently, we are minimizing the number of time period gaps at the excavation sites.

Application 38. Assembling physical mapping in genetics [Alizadeh, Karp, Newberg & Weisser, 1992; Arratia, Lander, Tivaré & Waterman, 1991]

The human genome consists of 23 pairs of chromosomes. Each chromosome contains a single very long molecule of DNA, which is constructed as a linear arrangement of 4 distinct nucleic acids, each represented by a letter A, C, T or G. Thus a chromosome can be thought of as a long sequence of letters such

as ACTGACCTGGATTCG... Biologists estimate that in total all the human chromosomes contain 3 billion letters. As a pictorial image, if printed in a single line of type, a line containing all of the letters would be over 6,000 miles long.

DNA has often been described as the 'language of genetics' and the chromosomes might be thought of as a 'blueprint' for the human body. In order to decode the information stored in the chromosomes, the United States government has funded The Human Genome Project, whose goal is to determine the sequence of genes in each chromosome and ultimately to determine the sequence of nucleic acids that makes up the DNA of each chromosome. An interim goal of the Human Genome Project is to create rough maps of the chromosomes known as 'physical maps.' Here we show how to construct a reasonably accurate physical map by solving a related traveling salesman problem.

A clone is a fragment of DNA, stored in such a way that copies are easy to make. A common method of storing extremely large clones (large in terms of the amount of DNA of the clone) is to store them inside a yeast cell. When the yeast cell duplicates, the clone also duplicates. By controlling the replication of the yeast, biologist can create as many copies of a clone as they want.

Biologists often refer to a collection of clones from the same DNA source as a library. Suppose for example that we have a library of chromosome number 1 from humans. Such a library is a very useful source of genetic material for biologists studying chromosome 1 (and perhaps looking for some genetic disease); unfortunately, in creating a library from a genetic source such as a human chromosome, we lose all positional information. The biologist has no simple way of determining where each clone lies on the chromosome. Rather, through experimental means, the scientists infer where clones lie; a physical map is a plausible (and hopefully accurate) layout of the clones on the chromosome.

A *probe* may be thought of as a unique point on the genome. Biologists can select probes randomly and they can experimentally determine whether a clone contains a probe. Let $a_{ij} = 1$ if clone j contains probe i, $a_{ij} = 0$ otherwise. As before, we insert a dummy probe n with $a_{nj} = 0$ for all j, that is, this 'artificial' probe does not occur in any clone. Assuming that probes occur uniquely within the genome, and assuming that experimental data is error free, then any two clones that have a common probe must overlap in the genome. Moreover, if the probes are arranged in the correct linear order on the chromosome, then the collection of probes contained by a given clone will occur in consecutive positions. Determining the order of the probes on the genome will simultaneously order the clones as well. Such an ordering of either the clones or the probes is referred to as a physical map. Figure 33 gives an example of some probe-clone data and a physical map.

Reordering the probes so that the probes within each clone are consecutive is equivalent to ordering the rows of A so that the 1's in each column are consecutive. This matrix ordering problem is known as 'determining whether A has the consecutive 1's property in columns' and is efficiently solvable.

Unfortunately, in practice, the data is not so perfect. Many clones contain not one but two fragments of DNA that were accidentally 'glued' together in the

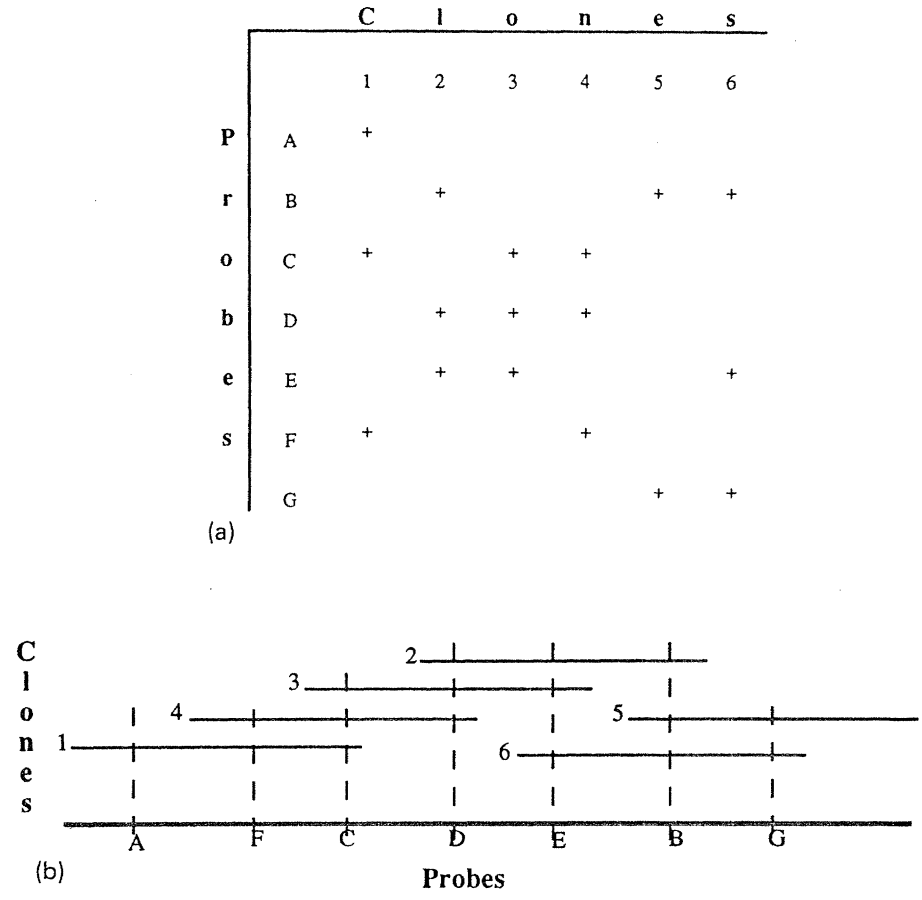


Fig. 33. (a) A '+' in entry (i, j) indicates that clone j contains probe i. (b) A physical map based on the data from (a).

cloning process. Two-fragment clones are referred to as chimeras. In addition, the data is subject to experimental error. Quite frequently an experiment falsely reports that a probe is not contained within a clone. (Error rates of 10% or higher are common.) Occasionally, an experiment also falsely reports that a probe is contained within a clone. To account for the problems with cloning and with experimental errors, an alternative objective is to order the rows of A in a way that minimizes the number of blocks of consecutive 1's, summed over all the columns. This is the matrix-represented traveling salesman problem in which the vector associated with the ith probe is A_i. In the case of perfect data, this problem reduces to the previous problem of identifying consecutive ones since each column will have exactly one block of consecutive ones.

Application 39. Optimal vane placement in turbine engines [Plante, Lowe & Chandresekaran, 1987]

A nozzle guide vane is a critical component of a gas turbine engine used in both military and commercial aircraft. The nozzle guide vane accelerates, deflects,

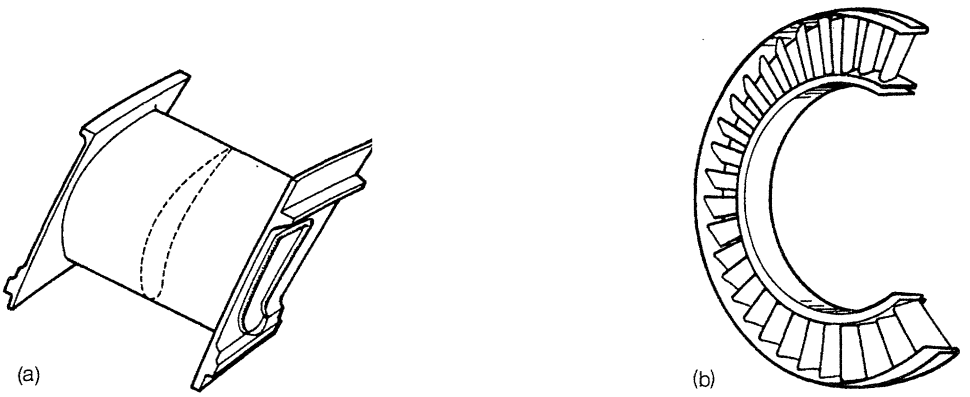


Fig. 34. Optimal vane placement in turbine engines. (a) Nozzle guide vane. (b) Nozzle guide assembly.

and distributes the gases that drive the turbine rotor. The nozzle assembly (see Figure 34) consists of a number of vanes (generally 46 to 100) placed around the circumference of the nozzle guide vane. Vane i has two associated parameters A_i and B_i that can be effectively measured. If vane j is placed immediately clockwise in relation to vane i, then the area between these two vanes is estimated to be $A_i + B_j$. Ideally, all of these areas should be approximately the same value, say d. A good choice in practice for ordering the vanes is a tour t that minimizes

$$(d - A_{t(n)} - B_{t(1)})^2 + \sum_{i=1}^{n-1} (d - A_{t(i)} - B_{t(i+1)})^2.$$

This order places the vanes so as to minimize the squared deviations from the target value d.

This problem is a special case of the traveling salesman problem, with $c_{ij} = (d - A_i - B_j)^2$. Observe that minimizing the displayed quantities is equivalent to minimizing $A_{t(n)}B_{t(1)} + \sum A_{t(i)}B_{t(i+1)}$ since the other terms in the expansion, $nd^2 - nd\sum A_{t(i)} - n\sum B_{t(i)} + \sum(A_{t(i)})^2 + \sum(B_{t(i)})^2$, are independent of the permutation. In this instance, the traveling salesman problem has the additional property that $c_{ij} = A_i B_j$. This special case of the traveling salesman problem is known as the product form, and it too is NP-hard.

Additional applications

The traveling salesman problem has also been applied to *cutting wallpaper* [Garfinkel, 1977], *job sequencing* [Gilmore & Gomory, 1964], *computer wiring* [Lenstra & Rinnooy Kan, 1975], *reading records from a rotating storage device* [Fuller, 1972], *clustering and aggregation* [McCormick, Schweitzer & White, 1972; Lenstra & Rinnooy Kan, 1975], *order picking in a warehouse* [Ratliff & Rosenthal, 1983].

13. Network design

In traditional network flow models (shortest paths, minimum cost and maximum flows, generalized flows, and multicommodity flows), we are given a network $G = (N, A)$ and we wish to route flow on this network in order to fulfill some underlying flow requirements. In network design, we are given a node set N and a set $\overline{A}$ of candidate arcs, and we wish to simultaneously define the underlying network $G = (N, A)$ with $A \subseteq \overline{A}$, and determine an optimal flow on it. In one standard version of this problem, we incur a fixed cost F_{ij} for including any arc (i, j) from A in the network.

The following model is an optimization formulation of the network design problem:

$$\text{minimize} \quad \sum_{k=1}^{K} \sum_{(i,j)\in\overline{A}} c_{ij}^k f_{ij}^k + \sum_{(i,j)\in\overline{A}} F_{ij} y_{ij} \tag{10a}$$

subject to

$$\sum_{j\in N} f_{ij}^k - \sum_{j\in N} f_{ji}^k = \begin{cases} 1, & \text{if } i = O(k) \\ -1, & \text{if } i = D(k) \\ 0, & \text{otherwise} \end{cases} \quad \text{for all } 1 \le k \le K \tag{10b}$$

$$\sum_{k=1}^{K} r_k f_{ij}^k \le u_{ij} y_{ij} \quad \text{for all } (i, j) \in \overline{A} \tag{10c}$$

$$f_{ij}^k \le y_{ij} \quad \text{for all } (i, j) \in \overline{A} \text{ and all } 1 \le k \le K \tag{10d}$$

$$f_{ij}^k \ge 0 \quad \text{for all } (i, j) \in \overline{A} \text{ and all } 1 \le k \le K \tag{10e}$$

$$y \in Y \tag{10f}$$

$$y_{ij} \ge 0 \quad \text{and integer for all arcs } (i, j). \tag{10g}$$

In this multicommodity flow version of the problem, each commodity $k = 1, 2, \ldots, K$, has an origin node $O(k)$, a destination node $D(k)$, and a flow requirement r_k. f_{ij}^k is the fraction of commodity k's flow requirement that flows on arc (i, j), and y_{ij} is an integer variable indicating how many copies of arc (i, j) we install on the network, each copy providing u_{ij} units of capacity. Another popular model, a 0–1 model, imposes the restriction $y_{ij} \le 1$ stating that we can add each arc at most once. Constraints (10b) are mass balance constraints. Constraints (10c) state that the total flow on arc (i, j) cannot exceed its installed capacity. Constraint (10d) is a forcing constraint stating that we can't flow on arc (i, j) if we don't install it, e.g., if $y_{ij} = 0$; this constraint is redundant in the integer programming version of this model, but not in its linear programming relaxation and so it is useful to include it in the model. Constraint (10f) permits us to impose restrictions on the design variables y_{ij}, for example, degree constraints on the

nodes. F_{ij} is the fixed cost for installing any copy of arc (i, j) and c_{ij}^k is the cost for routing all of the flow requirement of commodity k on arc (i, j).

This model in quite general. If we choose the data appropriately, it includes the following problems as special cases:

(1) *The minimum spanning tree problem*: If d_{ij} are the minimum spanning tree costs, then we set each $c_{ij}^k = 0$ and each $F_{ij} = F_{ji} = d_{ij}$ (note that in the network design model, the arcs are directed, so we replace each arc in the minimum spanning tree graph with two directed arcs, each with the same fixed cost); $u_{ij} > \sum_{k=1}^{K} r_k$, each $O(k)$ is the same node (say, node 1), Y states that the design has $|N| - 1$ arcs, and $r_k = 1$ for each node $k \neq 1$. Note that if each $d_{ij} > 0$, then we can eliminate constraints (10f) entirely and every optimal solution to the network design problem will be a spanning tree.

(2) *The traveling salesman problem*: If d_{ij} are the TSP costs, then we set each $c_{ij}^k = 0$ and each $F_{ij} = d_{ij}$; Y states that the design has $|N|$ arcs, each pair of nodes defines a commodity and each $r_k = 1$. Alternatively, if $F_{ij} = d_{ij} + M$ (M is a large constant), then we can eliminate the constraints Y and every optimal solution will be an optimal TSP tour.

(3) *Facility location models*: In this case, we use the familiar node splitting device of network flows to convert the decision as to whether we place a facility on a node into a decision about placing a facility on an arc.

We next describe several network design applications.

Application 40. Designing fixed cost communication and transportation systems

The primary purpose of a communication (transportation) system, such as telephone or computer networks, is to provide communication (transportation) links between the system's users. In many of these application contexts, routing costs are zero or negligible. Therefore, once we have constructed the system, we can send as many messages (trucks, passengers) as we like (within the system's operating capacity) at no cost. There are many variants on this general theme: for example, models with or without capacity restrictions, models with or without reliability considerations, and models that impose special topological restrictions (e.g., node degree constraints) on the system's design. The following generic models illustrate a few of these features. In each case, the only costs in the model are the fixed costs imposed upon the arcs.

Minimum spanning tree. If we need to connect all the nodes of the network, then we need to find a minimum spanning tree on the given network. As we have previously indicated, this model is a special case of the network design problem.

Steiner tree [Maculan, 1987; Winter, 1987]. If we need only a subset of the nodes of the network (customer nodes), then we need to find a minimum cost spanning tree that includes these nodes as well perhaps as some or all of the other nodes (called *Steiner nodes*). This generalization of the minimum spanning tree is known as the *Steiner tree problem*. The formulation of this model as a special case of the

network design model is similar to that of the minimum spanning tree except that now we assume node 1 is one of the customer nodes and we define commodities only for the customer nodes $k \neq 1$; for each such node k, we set $O(k) = 1$, $D(k) = k$ and $r_k = 1$.

Private network leasing [Magnanti, Mirchandani & Vachani, 1991]. In this instance, one or more types of facilities (communication lines) are available to carry flow on each arc: for example, DS0 and DS1 lines (digital service type 0 and 1 lines). DS1 lines cost more than DS0 lines, but have 24 times the flow carrying capacity. Given demands between the users of the system, we need to find a minimum cost network configuration that has sufficient capacity to carry the flow. Therefore, we need to specify how many units of each type of transmission line to load on each arc of the network and how to route the flow through the resulting network. This model is a variant of the network design model, with zero routing costs for all arcs and with two different types of parallel arcs joining each arc in A, a DSO arc and a DS1 arc. If we load x_{ij} units of DS0 lines and y_{ij} units of DS1 lines between nodes i and j, then one of these parallel arcs has a capacity x_{ij} and the other a capacity of $24y_{ij}$. (Therefore, the capacities on the righthand side of the constraints (10c) will be $u_{ij} = 1$ and $u_{ij} = 24$.)

In a freight transportation system, the arcs correspond to different types of trucks, for example 24 and 36 footers, each with a different capacity and cost, that we can dispatch on the arc.

Network survivability [Monma & Shallcross, 1989; Groetschel, Monma & Stoer, 1992]. Almost all networks are subject to failure, in the sense that periodically the nodes and/or arcs fail and therefore become unavailable to carry flow. With the advent of new high capacity transmission lines, for example, fiber optic lines, the failure of a single line can lead to enormous service disruptions (major 'brown outs'). To deal with these contingencies, network planners sometimes need to build redundancy into a network's deign. That is, rather than designing a spanning tree with a single path between each node, we might impose a redundancy or survivability requirement by specifying the need to include at least r_{ij} arc-disjoint paths (if the arcs are subject to failure) between nodes i and j. For this application, each pair of nodes i and j with $r_{ij} > 0$ defines a commodity. In this version of the 0–1 network design model, the routing costs are again zero, the capacity u_{ij} for each arc (i, j) is one, and each variable is a 0–1 variable.

Application 41. Local access telephone network capacity expansion [Balakrishnan, Magnanti & Wong, 1991; Balakrishnan, Magnanti, Shulman & Wong, 1991]

As the demand for services (voice, data, video) increases, telephone companies must determine the optimal investment policy for adding capacity to their networks. The local access component of the overall telephone system is the part of the service network that connects end subscribers to local switching centers and to the long distance, or backbone, network. Typically, the local access network uses

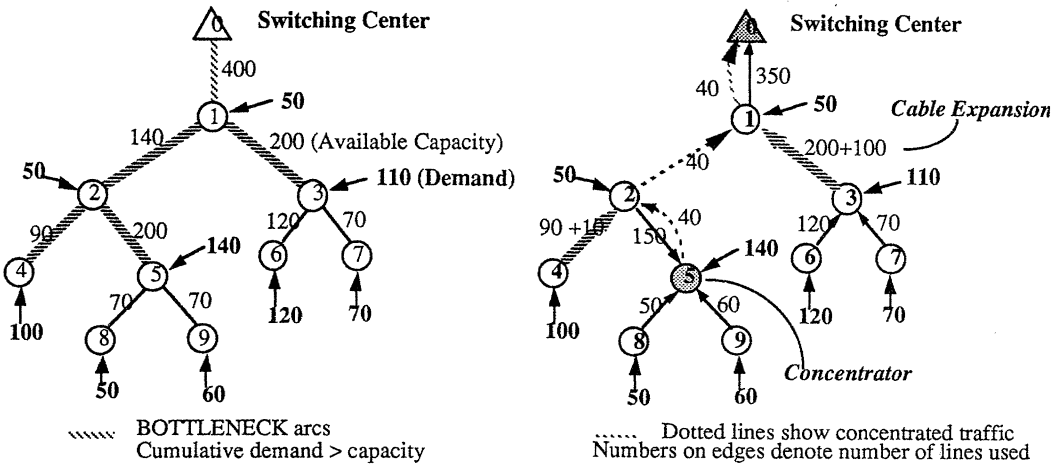


Fig. 35. Capacity expansion example. (a) Problem data; (b) expansion plan.

copper cables to carry messages and network has a tree structure as shown in Figure 35a. In this example, the current capacity of links (0,1), (1,2), (1,3), (2,4) and (2,5) are insufficient to meet future demand.

Telephone companies have several options for adding capacity. They can (i) add more copper cable to any link of the network, (ii) add fiber optics links to the network, and (iii) add remote switches and message concentrators to the nodes of the network. Figure 35b shows a possible expansion plan for this problem (without the fiber optic cable option). This expansion loads the network with additional capacity on links (1,3) and (2,4) and adds a concentrator (with a compression ratio of 10) at node 5 which handles all the traffic from nodes 2, 5, 8, and 9. The concentrator makes it possible to compress messages so that they use less capacity on all arcs that lie 'upstream' towards the local switch.

In one popular practical version of this problem, (i) all the traffic from any subscriber node i must home to (i.e., be routed to) the same node j which must contain a concentrator or the local switch, and (ii) the traffic pattern must satisfy a *contiguity property* stating that if node i homes to node j and node k lies on the tree path P_{ij} between nodes i and j, then node k must also home to node j. Therefore, any feasible solution is a 'packing' of subtrees in the given tree T; that is, is a collection of subtrees $T_1, T_2, \ldots, T_q$ of T that are node-disjoint and that collectively contain all the nodes of T. In Figure 35b, the tree T_1 contains the nodes 2, 4, 5, 8, and 9 and the tree T_2 contains the nodes 1, 3, 6, and 7. We assume that we either connect any concentrator directly to the switching center or that the compression ratio of any concentrator is so high that we can ignore the capacity used by any compressed traffic. Therefore, the cost of any feasible solution is just the sum of costs of the subtrees $T_1, T_2, \ldots, T_q$. Then the cost of the subtree is (i) the fixed cost F_j of placing a concentrator at some node j in the subtree, (ii) the cost a_{ij} for assigning any node i in the tree to node j (this is the variable cost of adding more capacity to a concentrator), and (iii) the cost, if any, of expanding any arc of the network beyond its current capacity

B_{ij} so that it has sufficient capacity to carry the required flow in the subtree to node j.

Letting node 0 be the location of the switching center in the given local access tree, we can view this problem is a variant of the network design model (10) if our underlying network has three type of arcs: (i) an arc for each arc (i, j) in the tree with $u_{ij} = B_{ij}$ and with no associated fixed or routing costs, (ii) a parallel arc to (i, j) with the fixed cost of cable expansion on that arc and a routing cost equal to the variable cable expansion cost, and (iii) an arc $(j, 0)$ for each node j in T with fixed and routing costs equal to the fixed and variable costs for installing a concentrator at that node.

In the special instance of this problem without any existing capacity on any arc, the problem is easy to solve by dynamic programming [see Aghezzaf, Magnati & Wolsey, 1992; Barany, Edmonds & Wolsey, 1986]. In this case, we can include the arc expansion cost for assigning node i to node j (that is, the cost of expanding all the arcs on the path P_{ij}) in the assignment cost a_{ij}, and so the problem has only the costs (i) and (ii). In the general case, the model will contain added complexity (this version of the problem is NP-hard), but can be solved fairly efficiently by a Lagrangian relaxation decomposition method that solves the version of the problem without any existing arc capacities as a subproblem. This modeling and solution approach extends to more general situations as well – for example, a version of the problem with alternate types of concentrators, each with its own fixed and variable costs.

Application 42. Multi-item production planning [Leung, Magnati & Vachani, 1989; Pochet & Wolsey, 1991]

In Application 3, we showed how to convert certain single-item production planning problems into shortest path and minimum cost flow problems. In more general problem settings with production capacities or with multiple items that share scarce resources (e.g., machine capacity), the problems often become network design models. Even the special case that we examined in Application 3 is a special network design problem with the following interpretation (see Figure 2): (i) the production arcs $(0, t)$ for $t = 1, 2, \ldots, T$ incur a fixed cost F_t; (ii) we can define a commodity t for each time period t with $O(t) = 1$, $D(t) = t$, and $r_t = d_t$; (iii) the routing cost of the arcs $(0, t)$ equals the variable production cost c_t; (iv) the arcs $(t, t + 1)$ and $(t + 1, t)$ have zero fixed costs and have a routing cost for commodity k equal to the one period inventory holding cost and backorder cost for the entire demand d_k of that item. For the uncapacitated version of the problem that we considered in Application 3, $u_{ij} > \sum_{t=1}^{T} d_t$ and so we can eliminate the constraints (10c). We can also eliminate the constraints (10f).

When we impose production capacities, we include constraint (10c) for each arc $(0, t)$ with u_{0t} equal to the production capacity in period t (often $u_{0t} = U$, a constant capacity throughout the planning horizon).

For multi-item version of the problem with Q items, we can define a 'multi-layered' version of the network in Figure 3, with a demand node qt for each item

$q = 1, 2, \ldots, Q$, and time period t. We also insert a new node t in each period with an arc $(0, t)$ that carries the total production in period t, and arcs (t, qt) that carries the production in period t of item $q = 1, 2, \ldots, Q$. The flow on arc $(0, t)$ equals the sum of the flows on the arcs (t, qt) for $q = 1, 2, \ldots, Q$. For each commodity $O(q, t) = 1$, $D(q, t) = t$, and $r_{qt} = d_{qt}$, the demand for item q in period t. Arc $(0, t)$ imposes a total production capacity for all items in period t and the arcs (t, qt) impose a production capacity, if any, for item q in period t.

Other versions of this problem can also be viewed as network design problems, for example, in some applications we can produce only one item in any period and we incur a changeover cost whenever we switch between the production of any two items (for example, to clean dies or to reconfigure machine settings). Magnanti & Vachani [1990] and Van Hoesel, Wagelmans & Wolsey [1991] provide details about the network design formulation of these problems.

Additional applications

Magnanti & Wong [1984] and Minoux [1989] provide many additional references on network design, including several additional applications, and Magnanti & Wolsey [1993] describe applications, models and algorithms for several tree versions of network design problems. Bertsekas & Gallagar [1992], Graves, Rinnooy Kan & Zipkin [1993], and Mirchandani & Francis [1990] contain extensive discussions of communication networks, production planning, and facility location. Some additional applications of network design include (1) *airline route planning* [Jaillet & Yu, 1993]; (2) *centralized network design* [Gavish, 1984]; (3) *computer teleprocessing networks* [Chandy & Russell, 1972; Esau & Williams, 1966; Kershenbaum & Boorstyn, 1975]; (4) *electricity distribution planning* [Bousba & Wolsey, 1990; Gascon, Benchakroun & Ferland, 1991]; (5) *less-than truckload (LTL) consolidation* [Leung, Magnanti & Singhal, 1990; Braklow, Graham, Hassler, Peck & Powell, 1992]; (6) *regional water resources systems* [Jarvis, Rardin, Unger, Moore & Schimpeler, 1978]; and (7) *VLSI design* [Korte, Promel & Steger, 1990].

14. Summary

In this paper, we have described several applications of the following network optimization problems: shortest paths, maximum flows, minimum cost flows, assignment and matchings, minimum spanning trees, convex cost flows, generalized flows, multicommodity flows, the traveling salesman problem, and network design. We have adapted about 75% of these applications from Ahuja, Magnanti & Orlin [1993] who describe many other applications as well. The survey papers of Bennington [1974], Glover & Klingman [1976], Bodin, Golden, Assad & Ball [1983], Aronson [1989], and Glover, Klingman & Phillips [1990] provide many additional application references. The books by Gondran & Minoux [1984], Evans & Minieka [1992], and Glover, Klingman & Phillips [1992] also describe a variety of applications of network optimization.

Ahuja, R.K. (1986). Algorithms for the minimax transportation problem. *Nav. Res. Logistics Q.* 33, 725–740.

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

Ali, A.I., D. Barnett, K. Farhangian, J.L. Kennington, B. Patty, B. Shetty, B. McCarl and P. Wong (1984). Multicommodity network problems: Applications and computations. *AIIE Trans.* 16, 127–134.

Ali, A.I., R.V. Helgason and J.L. Kennington (1978). The convex cost network flow problem: A state-of-the-art survey. Technical Report OREM 78001, Southern Methodist University, Texas.

Alizadeh, F., R.M. Karp, L.A. Newberg and D.K. Weisser (1992). Physical mapping of chromosomes: A combinatorial problem in molecular biology. Unpublished manuscript.

Altinkemer, K., and B. Gavish (1991). Parallel savings based heuristics for the delivery problem, *Oper. Res.* 39, 456–469.

Anderson, W.N. (1975). Maximum matching and the rank of a matrix. *SIAM J. Appl. Math.* 28, 114–123.

Arisawa, S., and S.E. Elmaghraby (1977). The 'hub' and 'wheel' scheduling problems. *Transp. Sci.* 11, 124–146.

Aronson, J.E. (1989). A survey of dynamic network flows. *Ann. Oper. Res.* 20, 1–66.

Arratia, R., E.S. Lander, S. Tivaré and M.S. Waterman (1991). Genomic mapping by anchoring random clones: A mathematical analysis. *Genomics* 11, 806–827.

Assad, A.A. (1980). Models for rail transportation. *Transp. Res.* 14A, 205–220.

Bacharach, M. (1966). Matrix rounding problems. *Manage. Sci.* 9, 732–742.

Balakrishnan, A., T.L. Magnanti and R. Wong (1991). A decomposition algorithm for local access telecommunications network expansion planning. Working Paper, Operations Research Center, MIT.

Balakrishnan, A., T.L. Magnanti, A. Shulman and R.T. Wong (1991). Models for planning capacity expansion in local access telecommunication networks. *Ann. Oper. Res.* 33, 239–284.

Ball, M.O., L. Bodin and R. Dial (1983). A matching based heuristic for scheduling mass transit crews and vehicles. *Transp. Sci.* 17, 4–31.

Barany, I., J. Edmonds and L.A. Wolsey (1986). Packing and covering a tree by subtrees. *Combinatorica* 6, 245–257.

Barr, R.S., and J.S. Turner (1981). Microdata file merging through large scale network technology. *Math. Program. Study B* 15, 1–22.

Barros, O., and A. Weintraub (1986). Spatial market equilibrium problems as network models. *Discr. Appl. Math.* 13, 109–130.

Bartholdi, J.J., J.B. Orlin and H.D. Ratliff (1980). Cyclic scheduling via integer programs with circular ones. *Oper. Res.* 28, 1074–1085.

Barzilai, J., W.D. Cook and M. Kress (1986). A generalized network formulation of the pairwise comparison consensus ranking model. *Manage. Sci.* 32, 1007–1014.

Belford, P.C., and H.D. Ratliff (1972). A network-flow model for racially balancing schools. *Oper. Res.* 20, 619–628.

Bellman, R. (1958). On a routing problem. *Q. Appl. Math.* 16, 87–90.

Bellmore, M., G. Bennington and S. Lubore (1971). A multivehicle tanker scheduling problem. *Transp. Sci. B* 5, 36–47.

Bennington, G.E. (1974). Applying network analysis. *Ind. Eng.* 6, 17–25.

Bentley, J.L. (1990). Experiments on traveling salesman heuristics. *Procedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, Calif., pp. 91–99.

Berge, C., and A. Ghouila-Houri (1962). *Programming, Games and Transportation Networks*. John Wiley and Sons, Chichester.

Berrisford, H.G. (1960). The economic distribution of coal supplies in the gas industry: An application of the linear programming transport theory. *Oper. Res. Q.* 11, 139–150.

Bertsekas, D., and R. Gallagar (1992). *Data Networks* (2nd ed.), Prentice-Hall, Englewood Cliffs, N.J.

Some of the models included in this discussion have direct relevance to practitioners in such fields as communications, manufacturing and transportation; other models are generic mathematical problems of some interest to applied mathematicians (e.g., finding a solution to certain types of inequality systems) and that themselves have applications in varied practical settings (for example, in personnel scheduling). Some models are formulated naturally as network optimization problems; others don't appear naturally as network models, but are transformable into the form of network optimization problems (for example, by taking linear programming duals of 'natural' formulations). Taken as a whole, the models we have considered, and those we have cited, establish network optimization as an unusually powerful and rich modeling environment, adding evidence to justify the claim that applied mathematics, computer science, and operations research do indeed have much to offer to the world of practice.

Acknowledgments

The applications described in this paper are drawn from a number of operations research, computer science, and applied mathematics journals. The integer programming bibliographies by Kastning [1976], Hausman [1978] and Von Randow [1982, 1985] have been particularly valuable in helping us to identify these applications. Many of the applications we cover are excerpted from the book, *Network Flows: Theory, Algorithms, and Applications*, written by Ahuja, Magnanti & Orlin [1993]; this book describes over 150 applications of network optimization problems. We are indebted to Prentice-Hall for its permission to collect together and summarize these applications from our book in this chapter.

Figures 34a and 34b are reprinted by permission of the Operations Research Society of America and the authors of the paper 'The product matrix traveling salesman problem: An application and solution heuristic', *Operations Research* **35**, 772–783.

We are grateful to Professor Michael Ball for a number of valuable comments on an earlier version of this chapter, and, in particular, for pointing out two of the applications we have used.

The first and third authors have been supported by grants from the United Parcel Service, Prime Computers, and the Air Force Office of Scientific Research (Grant Number AFORS-88-0088).

References

Abdallaoui, G. (1987). Maintainability of a grade structure as a transportation problem. *J. Oper. Res. Soc.* 38, 367–369.

Aghezzaf, E.H., T.L. Magnanti and L.A. Wolsey (1992). Optimizing constrained subtrees of trees, CORE Discussion Paper No. 9250, Université Catholique de Louvain.

Ahlfeld, D.P., R.S. Dembo, J.M. Mulvey and S.A. Zenios (1987). Nonlinear programming on generalized networks. *ACM Trans. Math. Software* 13, 350–367.

Bousba, C., and L.A. Wolsey (1990). Finding minimum cost directed trees with demands and capacities. Discussion Paper No. 8913, Center for Oper. Res. and Econometrics, University of Louvain, Belgium.

Bodin, L.D., B.L. Golden, A.A. Assad and M.O. Ball (1983). Routing and scheduling of vehicles and crews: The state of the art. *Comput. Oper. Res.* 10, 69–211.

Braklow, J.W., W.W. Graham, S. Hassler, K.E. Peck and W.B. Powell (1992). Interactive optimization improves service and systems performance for yellow freight. *Interfaces* 22, 147–172.

Brogan, W.L. (1989). Algorithm for ranked assignments with application to multiobject tracking. *J. Guidance*, 357–364.

Cabot, A.V., R.L. Francis and M.A. Stary (1970). A network flow solution to a rectilinear distance facility location problem. *AIIE Trans. B* 2, 132–141.

Cahn, A.S. (1948). The warehouse problem (Abstract). *Bull. Am. Math. Soc.* 54, 1073.

Carraresi, P., and G. Gallo (1984). Network models for vehicle and crew scheduling. *Eur. J. Oper. Res.* 16, 139–151.

Chalmet, L.G., R.L. Francis and P.B. Saunders (1982). Network models for building evacuation. *Manage. Sci.* 28, 86–105.

Chandy, K.M., and R.A. Russell (1972). The design of multipoint linkages in a teleprocessing tree network. *IEEE Trans. Comput.* C-21, 1062–1066.

Cheshire, M., K.I.M. McKinnon and H.P. Williams (1984). The efficient allocation of private contractors to public works. *J. Oper. Res. Q.* 35, 705–709.

Clark, J.A., and N.A.J. Hastings (1977). Decision networks. *Oper. Res. Q.* 20, 51–68.

Clarke, S., and J. Surkis (1968). An operations research approach to racial desegregation of school systems. *Socio-Economic Planning Sci.* 1, 259–272.

Collins, M., L. Cooper, R. Helgason, J. Kennington and L. Leblanc (1978). Solving the pipe network analysis problem using optimization techniques. *Manage. Sci.* 24, 747–760.

Cox, L.H., and L.R. Ernst (1982). Controlled rounding. *INFOR* 20, 423–432.

Cormen, T.H., C.L. Leiserson, and R.L. Rivest (1990). *Introduction to Algorithms* MIT Press and McGraw-Hill, New York.

Crainic, T., J.A. Ferland and J.M. Rousseau (1984). A tactical planning model for rail freight transportation. *Transp. Sci.* 18, 165–184.

Crum, R.L., and D.J. Nye (1981). A network model of insurance company cash flow management. *Math. Program.* 15, 86–101.

Dafermos, S., and A. Nagurney (1984). A network formulation of market equilibrium problems and variational inequalities. *Oper. Res. Lett.* 5, 247–250.

Daniel, R.C. (1973). Phasing out capital equipment. *Oper. Res. Q.* 24, 113–116.

Dantzig, G.B. (1962). *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ.

Dantzig, G.B., and D.R. Fulkerson (1954). Minimizing the number of tankers to meet a fixed schedule. *Nav. Res. Logistics Q.* 1, 217–222.

Dearing, P.M., and R.L. Francis (1974). A network flow solution to a multifacility minimax location problem involving rectilinear distances. *Transp. Sci.* 8, 126–141.

Dembo, R.S., J.M. Mulvey and S.A. Zenios (1989). Large-scale nonlinear network models and their applications. *Oper. Res. B* 37, 353–372.

Denardo, E.V., U.G. Rothblum and A.J. Swersey (1988). A transportation problem in which costs depend on the order of arrival. *Manage. Sci.* 34, 774–783.

Derman, C., and M. Klein (1959). A note on the optimal depletion of inventory. *Manage. Sci.* 5, 210–214.

Devine, M.V. (1973). A model for minimizing the cost of drilling dual completion oil wells. *Manage. Sci.* 20, 532–535.

Dewar, M.S.J., and H.C. Longuet-Higgins (1952). The correspondence between the resonance and molecular orbital theories. *Proc. R. Soc. London B* A214, 482–493.

Dirickx, Y.M.I., and L.P. Jennergren (1975). An analysis of the parking situation in the downtown area of West Berlin. *Transp. Res.* 9, 1–11.

Dorsey, R.C., T.J. Hodgson and H.D. Ratliff (1975). A network approach to a multifacility multi-product production scheduling problem without backordering. *Manage. Sci.* 21, 813–822

Dress, A.W.M., and T.F. Havel (1988). Shortest path problems and molecular conformation. *Discr. Appl. Math.* 19, 129–144.

Dror, M., P. Trudeau and S.P. Ladany (1988). Network models for seat allocation on flights. *Transp. Res.* 22B, 239–250.

Dude, R.O., and P.E. Hart (1973). *Pattern Classification and Science Analysis*. Wiley-Interscience, New York, N.Y.

Edmonds, J. (1967). An introduction to matching. Mimeographed notes, Engineering Summer Conference, The University of Michigan, Ann Arbor.

Edmonds, J., and E.L. Johnson (1973). Matching, Euler tours and the Chinese postman. *Math. Program.* 5, 88–124.

Elmaghraby, S.E. (1978). *Activity Networks: Project Planning and Control by Network Models*. Wiley-Interscience, New York, N.Y.

Esau, L.R., and K.C. Williams (1966). On teleprocessing system design II. *IBM Systems J., B* 5, 142–147.

Evans, J.R. (1977). Some network flow models and heuristics for multiproduct production and inventory planning. *AIIE Trans.* 9, 75–81.

Evans, J.R. (1984). The factored transportation problem. *Manage. Sci.* 30, 1021–1024.

Evans, J.R., and E. Minieka (1992). *Optimization Algorithms for Networks and Graphs*. Marcel Dekker, Inc., New York, N.Y.

Ewashko, T.A., and R.C. Dudding (1971). Application of Kuhn's Hungarian assignment algorithm to posting servicemen. *Oper. Res. B* 19, 991.

Farley, A.R. (1980). Levelling terrain trees: A transshipment problem. *Inf. Process. Lett. B* 10, 189–192.

Federgruen, A., and H. Groenevelt (1986). Preemptive scheduling on uniform machines by network flow techniques. *Manage. Sci.* 32, 341–349.

Filliben, J.J., K. Kafadar and D.R. Shier (1983). Testing for homogeneity of two-dimensional surfaces. *Math. Modelling* 4, 167–189.

Florian, M., and D.W. Hearn (1992). Network equilibrium: Models and algorithms, to appear.

Ford, L.R., and D.R. Fulkerson (1958). Constructing maximal dynamic flows from static flows. *Oper. Res.* 6, 419–433.

Ford, L.R., and S.M. Johnson (1959). A tournament problem. *Am. Math. Mon.* 66, 387–389.

Francis, R.L., and J.A. White (1976). *Facility Layout and Location*. Prentice-Hall, Englewood Cliffs, N.J.

Frank, C.R. (1965). A note on the assortment problem. *Manage. Sci.* 11, 724–726.

Fujii, M., T. Kasami and K. Ninomiya (1969). Optimal sequencing of two equivalent processors. *SIAM J. Appl. Math.* 17, 784–789. Erratum, same journal 18, 141.

Fulkerson, D.R. (1961). A network flow computation for project cost curve. *Manage. Sci. B* 7, 167–178.

Fulkerson, D.R. (1965). Upsets in a round robin tournaments. *Can. J. Math.* 17, 957–969.

Fulkerson, D.R. (1966). Flow networks and combinatorial operations research. *Am. Math. Mon.* 73, 115–138.

Fulkerson, D.R., and G.C. Harding (1977). Maximizing the minimum source–sink path subject to a budget constraint. *Math. Program.* 13, 116–118.

Fuller, S.H. (1972). An optimal drum scheduling algorithm. *IEEE Trans. Comput.* C-21, 1153–1165.

Gallo, G., M.D. Grigoriadis and R.E. Tarjan (1989). A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.* 18, 30–55.

Garey, M.S., and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

Garfinkel, R.S. (1977). Minimizing wallpaper waste part I: A class of traveling salesman problems. *Oper. Res.* 25, 741–751.

Gascon, V., A. Benchakroun and J.A. Ferland (1991). Electricity distribution planning model: A

network design approach solving the master problem of the Benders decomposition method. Working Paper, University on Montreal.

Gavish, B. (1984). Augmented Lagrangean based algorithms for centralized network design. *IEEE Trans. Commun.* 33, 1247–1275.

Gavish, B., and P. Schweitzer (1974). An algorithm for combining truck trips. *Transp. Sci.* 8, 13–23.

Geoffrion, A.M., and G.W. Graves (1974). Multicommodity distribution system design by Benders decomposition. *Manage. Sci. B* 20 , 822–844.

Gilmore, P.C., and R.E. Gomory (1964). Sequencing a one state-variable machine: A solvable case of the travelling salesman problem. *Oper. Res.* 12, 655–679.

Glover, F., and D. Klingman (1976). Network applications in industry and government. *AIIE Trans. B* 9, 363–376.

Glover, F., and J. Rogozinski (1982). Resort development: A network-related model for optimizing sites and visits. *J. Leisure Res.*, 235–247.

Glover, F., D. Klingman and N. Phillips (1990). Netform modeling and applications. *Interfaces* 20, 7–27.

Glover, F., D. Klingman and N. Phillips (1992). *Network Models in Optimization and Their Applications*. John Wiley and Sons, Chichester.

Glover, F., J. Hultz, D. Klingman and J. Stutz (1978). Generalized networks: A fundamental computer based planning tool. *Manage. Sci.* 24, 1209–1220.

Glover, F., R. Glover and F.K. Martinson (1984). A netform system for resource planning in the U.S. Bureau of land management. *J. Oper. Res. Soc.* 35, 605–616.

Goetschalckx, M., and H.D. Ratliff (1988). Order picking in an aisle. *IIE Trans. B* 20, 53–62.

Golden, B.L. (1975). A minimum cost multicommodity network flow problem concerning imports and exports. *Networks* 5, 331–356.

Goldman, A.J., and G.L. Nemhauser (1967). A transport improvement problem transformable to a best-path problem. *Transp. Sci.* 1, 295–307.

Golitschek, M.V., and H. Schneider (1984). Applications of shortest path algorithms to matrix scalings. *Numer. Math. B* 44, 111–126.

Gondran, M., and M. Minoux (1984). *Graphs and Algorithms*. Wiley-Interscience, New York, N.Y.

Gorham, W. (1963). An application of a network flow model to personnel planning. *IEEE Trans. Eng. Manage.* 10, 113–123.

Gower, J.C., and G.J.S. Ross (1969). Minimum spanning trees and single linkage cluster analysis. *Appl. Stat.* 18, 54–64.

Groetschel, M., C.L. Monma and M. Stoer (1992). Design of surviavable networks, to appear.

Graham, R.L., and P. Hell (1985). On the history of minimum spanning tree problem. *Ann. Hist. Comput.* 7, 43–57.

Graves, S., A.H.G. Rinnooy Kan and P. Zipkin (eds.) (1993). In *Handbooks of Oper. Res. and Manage. Sci.*, Volume: *Logistics of Production and Inventory*, North-Holland.

Gupta, S.K. (1985). *Linear Programming and Network Models*. Affiliated East–West Press Private Limited, New Delhi.

Gusfield, D. (1988). A graph theoretic approach to statistical data security. *SIAM J. Comput* 17, 552–571.

Gusfield, D., and C. Martel (1992). A fast algorithm for the generalized parametric minimum cut problem and applications. *Algorithmica* 7, 499–519.

Gusfield, D., C. Martel and D. Fernandez-Baca (1987). Fast algorithms for bipartite network flow. *SIAM J. Comput.* 16, 237–251.

Gutjahr, A.L., and G.L. Nemhauser (1964). An algorithm for the line balancing problem. *Manage. Sci.* 11, 308–315.

Hall, M. (1956). An algorithm for distinct representatives. *Am. Math. Mon.* 63, 716–717.

Hausman, H. (1978). *Integer Programming and Related Areas: A Classified Bibliography*. Lecture Notes in Economics and Mathematical Systems, Vol. 160, Springer-Verlag, Berlin.

Haymond, R.E., J.R. Thornton and D.D. Warner (1988). A shortest path algorithm in robotics and its implementation on the FPS T-20 hypercube. *Ann. Oper. Res. B* 14, 305–320.

Held, M., and R. Karp (1970). The traveling salesman problem and minimum spanning trees. *Oper. Res.* 18, 1138–1162.

Hoffman, A.J., and H.M. Markowitz (1963). A note on shortest path, assignment and transportation problems. *Nav. Res. Logistics Q. B* 10, 375–379.

Hoffman, A.J., and S.T. McCormick (1984). A fast algorithm that makes matrices optimally sparse, in: *Progress in Combinatorial Optimization*, Academic Press.

Horn, W.A. (1971). Determining optimal container inventory and routing. *Transp. Sci.* 5, 225–231.

Horn, W.A. (1973). Minimizing average flow time with parallel machines. *Oper. Res.* 21, 846–847.

Hu, T.C. (1961). The maximum capacity route problem. *Oper. Res. B* 9, 898–900.

Hu, T.C. (1966). Minimum cost flows in convex cost networks. *Nav. Res. Logistics Q.* 13, 1–9.

Hu, T.C. (1967). Laplace's equation and network flows. *Oper. Res.* 15, 348–354.

Hunter, D. (1976). An upper bound for the probability of a union. *J. Appl. Probab. B* 13, 597–603.

Imai, H., and M. Iri (1986). Computational-geometric methods for polygonal approximations of a curve. *Comput. Vision, Graphics Image Process.* 36, 31–41.

Jaillet, P., and G. Yu (1993). An integer programming model for the airline network design problem, *Bulletin, National Meeting of the Oper. Res. Soc. Am. Inst. Manage. Sci.*, Phoenix, Arizona.

Jarvis, J.J., R.L. Rardin, V.E. Unger, R.W. Moore and C.C. Schimpeler (1978). Optimal design of regional wastewater systems: a fixed-charge network flow model, *Oper. Res.* 26, 538–550.

Jacobs, W.W. (1954). The caterer problem. *Nav. Res. Logistics Q. B* 1, 154–165.

Jewell, W.S. (1957). Warehousing and distribution of a seasonal product. *Nav. Res. Logistics Q. B* 4, 29–34.

Johnson, T.B. (1968). Optimum pit mine production scheduling. Technical Report, University of California, Berkeley.

Kang, A.N.C., R.C.T. Lee, C.L. Chang and S.K. Chang (1977). Storage reduction through minimal spanning trees and spanning forests. *IEEE Trans. Comput.* C-26, 425-434.

Kaplan, S. (1973). Readiness and the optimal redeployment of resources. *Nav. Res. Logistics Q.* 20, 625–638.

Kastning, C. (1976). *Integer Programming and Related Areas: A Classified Bibliography*. Lecture Notes in Economics and Mathematical Systems, Vol. 128, Springer-Verlag, Berlin.

Kelley, J.R., (1961). Critical path planning and scheduling: Mathematical basis. *Oper. Res. B* 9, 296–320.

Kelly, J.P., B.L. Golden and A.A. Assad (1992). Cell suppression: Disclosure protection for sensitive tabular data. *Networks* 22, 397–417.

Kennington, J.L., and R.V. Helgason (1980). *Algorithms for Network Programming*. Wiley-Interscience, New York, N.Y.

Kershenbaum, A., and R.B. Boorstyn (1975). Centralized teleprocessing network design, *Proc. National Telecommunications Conference*, 27.11–27.14.

Khan, M.R. (1979). A capacitated network formulation for manpower scheduling. *Ind. Manage. B* 21, 24–28.

Khan, M.R., and D.A. Lewis (1987). A network model for nursing staff scheduling. *Z. Oper. Res.* 31, B161–171.

Kolitz, S. (1991). Personal communication.

Korte, B. (1988). Applications of combinatorial optimization. Technical Report No. 88541-OR. Institute für Okonometrie und Operations Research, Nassestrasse 2, D-5300, Bonn.

Korte, B., H.J. Promel and A. Steger (1990). Steiner trees in VLSI layout, in: B. Korte, L. Lovasz, H.J. Promel, A. Schijver (eds.), *Paths, Flows and VLSI Layout*, Springer-Verlag, Berlin.

Kourtz, P. (1984). A network approach to least cost daily transfers of forest fire control resources. *INFOR B* 22, 283–290.

Larson, R.C., and A.R. Odoni (1981). *Urban Operations Research*. Prentice-Hall, Englewood Cliffs, N.J.

Lawania (1990). Personal communication.

Lawler, E.L. (1964). On scheduling problems with deferral costs. *Management Science* 11, 280–287.

Lawler, E.L. (1966). Optimal cycles in doubly weighted linear graphs. *Theory of Graphs: International Symposium,* Dunod, Paris and Gordon and Breach, New York, N.Y., pp. 209–213.

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

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds. (1985).. *The traveling salesman problem: A guided tour of combinatorial optimization.* John Wiley and Sons, Chichester.

Lenstra, J.K., and A.H.G. Rinnooy Kan (1975). Some simple applications of the travelling salesman problem. *Oper. Res. Q.* 26, 717–733.

Leung, J., T.L. Magnanti and V. Singhal (1990). Routing in point to point delivery systems. *Transp. Sci.* 24, 245–260.

Leung, J., T.L. Magnanti and R. Vachani (1989). Facets and algorithms for capacitated lot sizing. *Math. Program.* 45, 331–360.

Levner, E.V., and A.S. Nemirovsky (1991). A network flow algorithm for just-in-time project scheduling. Memorandum COSOR 91-21, Dept. of Mathematics and Computing Science, Eindhoven Univesity of Technology, Eindhoven.

Lin, T.F. (1986). A system of linear equations related to the transportation problem with application to probability theory. *Discr. Appl. Math.* 14, 47–56.

Love, R.R., and R.R. Vemuganti (1978). The single-plant mold allocation problem with capacity and changeover restriction. *Oper. Res. B* 26, 159–165.

Lowe, T.J., R.L. Francis and E.W. Reinhardt (1979). A greedy network flow algorithm for a warehouse leasing problem. *AIIE Trans.* 11, 170–182.

Luss, H. (1979). A capacity expansion model for two facilities. *Nav. Res. Logistics Q.* 26, 291–303.

Machol, R.E. (1961). An application of the assignment Problem. *Oper. Res. B* 9, 585–586.

Machol, R.E. (1970). An application of the assignment Problem. *Oper. Res. B* 18, 745–746.

Maculan, N. (1987). The steiner problem in graphs. *Ann. Discr. Math.* 31, 185–212.

Magirou, V.F. (1986). The efficient drilling of printed circuit boards. *Interfaces* 16, 13–23.

Magnanti, T.L. (1984). Models and algorithms for predicting urban traffic equilibria, in: M. Florian (ed.), *Transportation Planning Models,* North-Holland, Amsterdam, pp. 153–186.

Magnanti, T.L., P. Mirchandani and R. Vachani (1991). Modeling and solving the capacitated network loading problem. Working Paper, Oper. Res. Center, MIT.

Magnanti, T.L., and R. Vachani (1990). A strong cutting plane algorithm for production scheduling with changeover costs. *Oper. Res.* 38, 456–473.

Magnanti, T.L., and R. Wong (1984). Network design and transportation planning: Models and algorithms. *Transp. Sci.* 18, 1–55.

Magnanti, T.L., and L.A. Wolsey (1993). Optimal Trees, to appear.

Mamer, J.W., and S.A. Smith (1982). Optimizing field repair kits based on job completion rate. *Manage. Sci. B* 28, 1328–1334.

Manne, A.S. (1958). A target-assignment problem. *Oper. Res. B* 6, 346–351.

Martel, C. (1982). Preemptive scheduling with release times, deadlines and due times. *J. ACM* 29, 812–829.

Mason, A.J., and A.B. Philpott (1988). Pairing stereo speakers using matching algorithms. *Asia-Pacific J. Oper. Res.* 5, 101–116.

Maxwell, W.L., and R.C. Wilson (1981). Dynamic network flow modelling of fixed path material handling systems. *AIIE Trans.* 13, 12–21.

McCormick, W.T. Jr., P.J. Schweitzer and T.W. White (1972). Problem decomposition and data reorganization by a clustering technique. *Oper. Res.* 20, 993–1009.

McGinnis, L.F., and H.L.W. Nuttle (1978). The project coordinators problem. *OMEGA* 6, 325–330.

Minoux, M. (1989). Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks* 19, 313–360.

Mirchandani P.B., and R.L. Francis (1990). *Discrete Location Theory.* John Wiley and Sons, Chichester.

Monma, C.L., and D.F. Shallcross (1989). Methods for designing communication networks with certain two-connected survivability constraints. *Oper. Res. B* 37, 531–541.

Monma, C.L., and M. Segal (1982). A primal algorithm for finding minimum-cost flows in capacitated networks with applications. *Bell Systems Tech. J.* 61, 949–468.

Mulvey, J.M. (1979). Strategies in modeling: A personal scheduling example. *Interfaces* 9, 66–76.

Orlin, D. (1987). Optimal weapons allocation against layered defenses. *Nav. Res. Logistics B* 34, 605–617.

Orlin, J.B., and U.G. Rothblum (1985). Computing optimal scalings by parametric network algorithms. *Math. Program.* 32, 1–10.

Osteen, R.E., and P.P. Lin (1974). Picture skeletons based on eccentricities of points of minimum spanning trees. *SIAM J. Comput.* 3, 23–40.

Picard, J.C., and H.D. Ratliff (1973). Minimum cost cut equivalent networks. *Manage. Sci.* 19, 1087–1092.

Picard, J.C., and H.D. Ratliff (1978). A cut approach to the rectilinear distance facility location problem. *Oper. Res.* 26, 422–433.

Picard, J.C., and M. Queyranne (1982). Selected applications of minimum cuts in networks. *INFOR B* 20, 394–422.

Plante, R.D., T.J. Lowe and R. Chandresekaran (1987). The product matrix traveling salesman problem: An application and solution heuristic. *Oper. Res.* 35, 772–783.

Pochet, Y., and L.A. Wolsey (1991). Solving multi-item lot-sizing problems with strong cutting planes. *Manage. Sci.* 37, 53–67.

Prager, W. (1957). On warehousing problems. *Oper. Res.* 5, 504–512.

Prim, R.C. (1957). Shortest connection networks and some generalizations. *Bell Systems Tech. J.* 36, 1389–1401.

Ratliff, H.D. (1978). Network models for production scheduling problems with convex cost and batch processing. *AIIE Trans.* 10, 104–108.

Ratliff, H.D., and A.S. Rosenthal (1983). Order picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Oper. Res.* 31, 507–521.

Ravindran, A. (1971). On compact book storage in libraries. *Opsearch* 8, 245–252.

Rhys, J.M.W. (1970). A selection problem of shared fixed costs and network flows. *Manage. Sci.* 17, 200–207.

Sapountzis, C. (1984). Allocating blood to hospitals from a central blood bank. *Eur. J. Oper. Res.* 16, 157–162.

Schneider, M.H., and S.A. Zenios (1990). A comparative study of algorithms for matrix balancing. *Oper. Res.* 38, 439–455.

Schwartz, B.L. (1966). Possible winners in partially completed tournaments. *SIAM Rev.* 8, 302–308.

Schwartz, M., and T.E. Stern (1980). Routing techniques used in computer communication networks. *IEEE Trans. Commun.* COM-28, 539–552.

Segal, M. (1974). The operator-scheduling problem: A network flow approach. *Oper. Res.* 22, 808–824.

Sheffi, Y. (1985). *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods.* Prentice-Hall, Englewood Cliffs, N.J.

Servi, L.D. (1989). A network flow approach to a satellite scheduling problem. Research Report, GTE Laboratories, Inc., Waltham, MA.

Shier, D.R. (1982). Testing for homogeneity using minimum spanning trees. *The UMAP J.* 3, 273–283.

Slump, C.H., and J.J. Gerbrands (1982). A network flow approach to reconstruction of the left ventricle from two projections. *Comput Graphics Image Process.* 18, 18–36.

Srinivasan, V. (1974). A transshipment model for cash management decisions. *Manage. Sci.* 20, 1350–1363.

Srinivasan, V. (1979). Network models for estimating brand-specific effects in multiattribute marketing models. *Manage. Sci.* 25, 11–21.

Stillinger, F.H. (1967). Physical clusters, surface tension and critical phenomenon. *J. Chem. Phys. B* 47, 2513–2533.

Stone, H.S. (1977). Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Eng.* 3, 85–93.

Shuchat, A. (1984). Matrix and network models in archeology. *Math. Mag.* 57, 3–14.

Szadkowski. (1970). An approach to machining process optimization. *Int. J. Prod. Res. B* 9, 371–376.

Tso, M. (1986). Network flow models in image processing. *J. Oper. Res. Society* 37, 31–34.

Tso, M., P. Kleinschmidt, I. Mitterreiter and J. Graham (1991). An efficient transportation algorithm for automatic chromosome karyotyping. *Pattern Recognition Lett.* 12, 117–126.

Van Hoesel, C.P.M., A.P.M. Wagelmans and L.A. Wolsey (1991). Economic lot-sizing with start-up costs: The convex hull. Core Discussion Paper 9109, Universite Catholique de Louvain (to appear in *SIAM J. Discr. Math.*).

Van Slyke, R., and H. Frank (1972). Network reliability analysis: Part I. *Networks* 1, 279–290.

Veinott, A.F., and H.M. Wagner (1962). Optimal capacity scheduling — Part I and II. *Oper. Res.* 10, 518–547.

Von Randow, R. (1982). *Integer Programming and Related Areas: A Classified Bibliography 1978–1981.* Lecture Notes in Economics and Mathematical Systems, Vol. 197, Springer-Verlag, Berlin.

Von Randow, R. (1985). *Integer Programming and Related Areas: A Classified Bibliography 1981–1984.* Lecture Notes in Economics and Mathematical Systems, Vol. 243, Springer-Verlag, Berlin.

Wagner, D.K. (1990). Disjoint (s, t)-cuts in a network. *Networks* 20, 361–371.

Waterman, M.S. (1988). *Mathematical Methods for DNA Sequences.* CRC Press.

White, L.S. (1969). Shortest route models for the allocation of inspection effort on a production line. *Manage. Sci.* 15, 249–259.

White, W.W. (1972). Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers. *Networks B* 2, 211–230.

Winter, P. (1987). Steiner problem in networks: A survey. *Networks* 17, 129–167.

Worsley, R.J. (1982). An improved Bonferroni inequality. *Biometrika* 69, 297–302.

Wright, J.W. (1975). Reallocation of housing by use of network analysis. *Oper. Res. Q.* 26, 253–258.

Zahn, C.T. (1971). Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput.* C20, 68–86.

Zangwill, W.I. (1969). A backlogging model and a multi-echelon model of a dynamic economic lot size production system—A network approach. *Manage. Sci.* 15, 506–527.

Zawack, D.J., and G.L. Thompson (1987). A dynamic space–time network flow model for city traffic congestion. *Transp. Sci.* 21, 153–162.