Contents

Chapter 1

Network Definitions and Formulations

1.1 Introduction

Wanting to optimize is a basic human trait. We begin with an incident which illustrates the fact that the urge to optimize provides universal motivation.

When I joined the University of Michigan, we stayed in an apartment close to campus. Our daughter was three years old then, and we had a small poodle to which she was very much attached. One day the poodle was missing. We searched the entire neighborhood for it, but had no luck. Another day went by, but it did not return. Unable to console my tearful daughter, I sought the advice of a colleague. He suggested that I put an advertisement in the campus newspaper. They recommended the offer of a reward for the poodle's return. Everything was agreed and they began running the advertisement offering a reward of $150 for anyone who finds and returns the poodle. We were very confident that this would bring quick results. A week passed by, and I called the newspaper office to check the status. The girl who received the call recognized my voice and asked whether I was the professor with the missing poodle. I said yes, and asked to be connected to the advertising manager. She said that he was out. I then asked to be connected to his assistant. She said that he was out too. Then I

wanted to speak with the editor. She replied that he was out too. I remarked "Goodness! Is everyone out?" She replied, "Yes Professor, they are all out looking for your dog!"

In this story, all the people have an optimizing attitude which is almost universal. With many successful optimization algorithms, and modern digital computers for implementing them, many organizations are using them routinely to optimize their operations.

While constructing an optimization model for a system these days, the emphasis is on keeping it **computable** or **tractable**. The network models discussed in this book are among the most tractable. We have very efficient algorithms for solving them. Taking advantage of the special structure, they are able to solve very large problems many times faster than general purpose linear programming algorithms. Excellent computer implementations of these algorithms are widely available. In this book we discuss network algorithms in all their variety and depth. In this chapter we discuss the basic network definitions and present some formulation examples.

Networks, Nodes, Lines, Arcs, Edges, Subnetworks, Graphs

A **network** is a pair of sets $(\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is a set of **points** (also called **vertices** or **nodes**) and $\mathcal{A}$ is a set of **lines**, each line joining a pair of points, together with some associated data. Nodes i, j are said to be **adjacent** if there is a line joining them.

A line joining i, j that can only be used in the direction from i to j is called an **arc**, and denoted by the **ordered pair** (i, j) with a comma, it is **incident into** j and **out of** i , node i is its **tail** , and j is its **head**. If (i, j) is denoted by e, then $i = \text{tail}(e)$, $j = \text{head}(e)$. For example (2, 1) is an arc in the network in Figure 1.1 with tail 2, and head 1.

A line joining two points x, y that can be used either from x to y, or from y to x, is called an **edge** and denoted by the **unordered pair** $(x; y)$ with a semicolon, it is **incident** at x and y. For example (4; 2) is an edge in the network in Figure 1.1 incident at nodes 4 and 2.

There can be more than one arc with the same orientation, or more

than one edge joining points i, j; such lines are called **parallel lines**. They will then be denoted by $(i,j)_1, (i,j)_2$, etc. When the network has such parallel lines, each of them is considered a separate distinct line by itself. However, we assume, except in Chapter 8, that there are no **self loops**, which are lines joining a point with itself. Figure 1.1 is a network with 8 points, 16 arcs, and 7 edges. Arcs (3, 5) and (2, 6) intersect in it, but their point of intersection is not a point in the network. The network is a **directed network** if all the lines in it are arcs, an **undirected network** if all the lines are edges, and a **mixed network** if it has both arcs and edges.

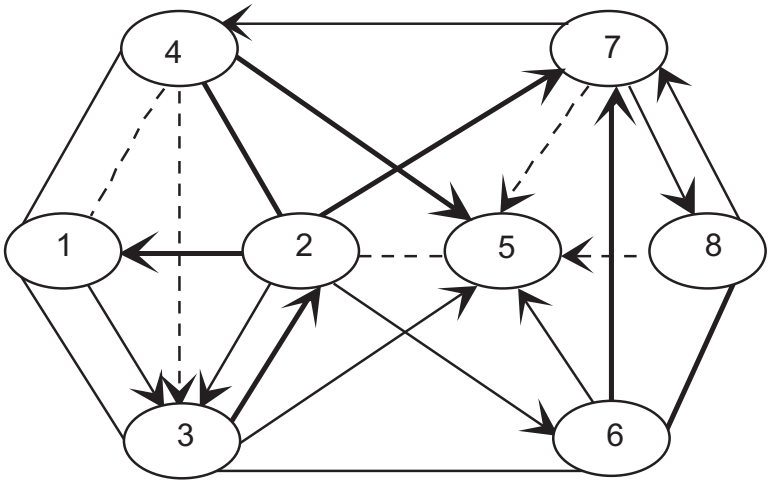


Figure 1.1:

The **degree** of a point in a network is the number of lines incident at it. In a directed network, the **indegree** (**outdegree**) of a node is the number of arcs incident into (out of) it.

A **subnetwork** of G $= (\mathcal{N}, \mathcal{A})$ is a network F $= (\mathcal{N}, \bar{\mathcal{A}})$ with the same set of points, but with $\bar{\mathcal{A}} \subset \mathcal{A}$.

A **partial network** of G is a network $(\hat{\mathcal{N}}, \hat{\mathcal{A}})$ in which the set of points is $\hat{\mathcal{N}} \subset \mathcal{N}$, and the set of lines is $\hat{\mathcal{A}}$ which is the set of all the lines in G that have both their incident points in $\hat{\mathcal{N}}$. This partial network is also called the **partial network** or **subnetwork of** G **induced by** the subset of nodes $\hat{\mathcal{N}}$. A **partial subnetwork** of G is a partial

network of a subnetwork of G. For example, by omitting all the thick and dashed lines and all the lines incident at node 8 in Figure 1.1, we get the subnetwork in Figure 1.2. The partial network of the network in Figure 1.1 induced by the subset of nodes {1, 2, 6, 8} is given in Figure 1.3 (a) (on the left). See Figure 1.3 (b) (on the right) for a partial subnetwork.

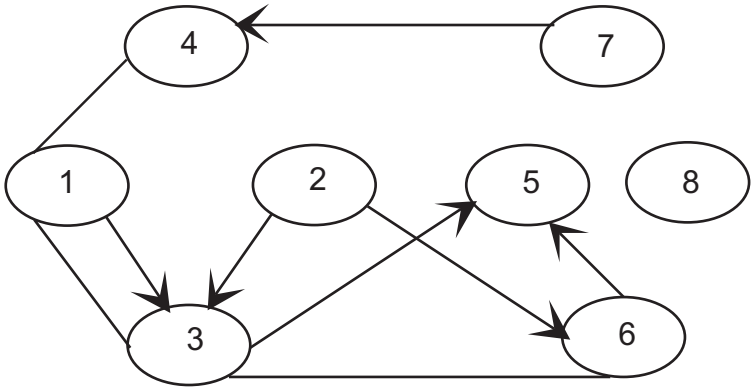
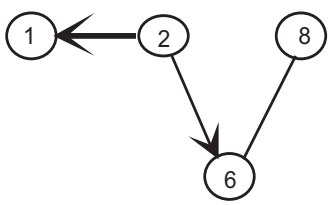
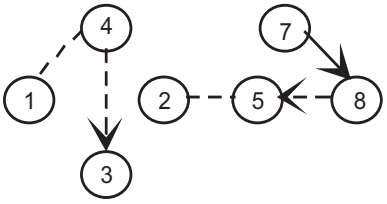


Figure 1.2: A subnetwork of the network in Figure 1.1.



(a) A partial network of the network in Figure 1.1.



(b) A partial subnetwork of the network in Figure 1.1.

Figure 1.3:

In a directed network, the **forward star** (**reverse star**) at a node i is the set of all arcs incident out of (incident into) i. It is sometimes convenient to represent directed networks by storing the forward stars of the nodes, or the reverse stars, or both. Searching through

forward (or reverse) stars of nodes is a common operation in many network algorithms, this representation is convenient for carrying out this search. In a directed network $\mathrm{G} = (\mathcal{N}, \mathcal{A})$, for $i \in \mathcal{N}$, the set $\mathbf{A}(i) = \{j : (i, j) \in \mathcal{A}\}$ is called the **after** i set, and $\mathbf{B}(i) = \{j : (j, i) \in \mathcal{A}\}$ is called the **before** i set.

Mathematically, an undirected network which has no self loops, is called a **graph** if it has no parallel edges, or a **multigraph** otherwise. A directed network which has no self loops is called a **digraph** if it has no parallel arcs, or a **multidigraph** otherwise.What distinguishes these graphs from networks is the fact that networks usually have some data such as arc lengths, or arc capacities associated with points and/or lines. Our problems come with data; hence we will use the term **network** to refer to our structures.

Application Areas

Network models are used extensively to analyze and optimize the operation of many systems.

For example, vehicular traffic in highway systems is studied using the network in which points are traffic centers and lines are roadway segments joining pairs of points.

Similarly, airline systems, railroad systems, shipping systems, and all transportation systems can be analyzed using appropriately defined networks.

Natural gas, crude oil, or other fluid flows are analyzed using the appropriate pipeline and/or ship or truck route networks.

The telephone network is the network for studying the flow of calls.

Economic models can be treated as network models by representing factories, warehouses, and markets as points and by treating highways, railroads, waterways, and other transportation channels as lines.

Even software and computer systems are analyzed using network models. A software system is often a collection of many components such as program modules, command procedures, data files, etc. The execution-time relationships and data communications between them form a directed network called the **call graph** of the program, nodes in it are procedures, and each arc represents one or more invocations

of a procedure by another. Similarly, distributed computer systems are analyzed using appropriate network models that represent their working modes.

Network models find applications in the design and analysis of many other types of systems, as most systems have to transmit goods or messages etc., through an appropriate transportation or communication medium. The application of network algorithms to solve these models benefits society by optimizing distribution, communication, and transportation costs, and improving productivity.

Also, being pictorial, a network model is visually informative and easy to construct and explain to top management.

Origin of Network Modeling, and Theory

The origin of network theory can be traced to the work of the famous Swiss mathematician Leonhard Euler on the **Königsberg bridges problem** in the year 1736.

The Königsberg bridges problem

The problem originated as a children's game in Königsberg on the banks of the river Pregel. There were two islands (denoted as land areas 1, 4; 2, 3 denote the two banks of the river) and seven bridges each joining a pair of land areas as in Figure 1.4. The question is: Is there a route beginning in one of the land areas, passing through each of the bridges exactly once, and returning to the initial land area at the end?

The town's children had long amused themselves running across the various bridges endlessly trying to find such a route, but no one succeeded.

It intrigued Euler, and when he grew up, he constructed a network model and developed an elegant method to answer this question. He represented each land area by a node, and each bridge by an edge joining the corresponding pair of nodes as in Figure 1.5. The desired route corresponds to one that begins at a node, traverses through each edge exactly once, and terminates at the starting node. Such a route

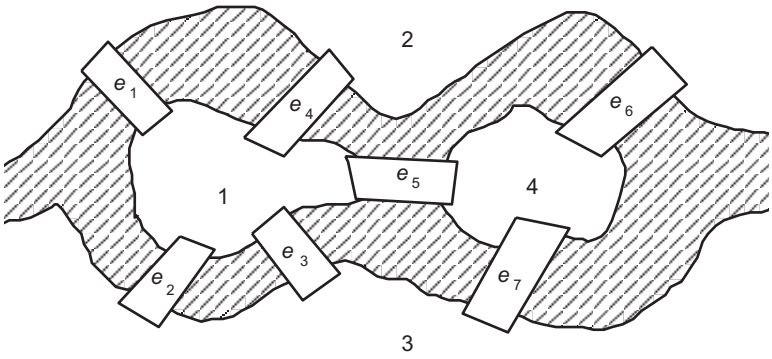


Figure 1.4: The Königsberg bridges.

is nowadays called an **Euler route** or **Euler circuit**. He proved that such a route exists in an undirected network iff it is connected (i.e., it is possible to pass from any node to any other node using the edges of the network), and every node has even degree (Theorem 1.11 of Section 1.3.8). Since the network in Figure 1.5 contains nodes which are not of even degree, the answer to the Königsberg bridges problem is: "no".

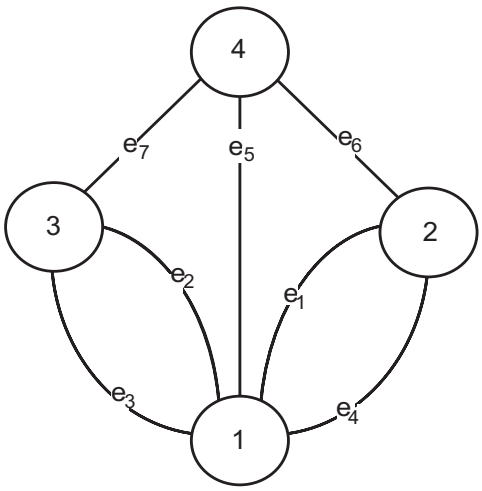


Figure 1.5: Network for the Königsberg bridges problem model.

Euler provided an elegant answer to an existence question. Sec-

tion 1.3.8 discusses efficient methods for actually computing an Euler route when it exists, and their application to solve important routing problems.

Problems, Algorithms, Computational Complexity

All the problems that we discuss in this book come with input data. The word **problem** normally refers to the general question to be answered, stated using mathematical symbols to represent the input data. An **instance** of this problem is obtained by providing specific values for all the input data symbols. For example, **the maximum value flow problem** is the problem of finding a maximum value flow in the single commodity flow network $G = (\mathcal{N}, \mathcal{A}, \ell, k, \breve{s}, \breve{t})$ (ℓ, k, $\breve{s}$, $\breve{t}$ are data, their meanings are explained later). Finding the maximum value flow in the specific network with data given in Figure 2.8 in Chapter 2, is an instance of this problem.

Measures of the largeness of a problem instance are provided by its parameters such as its **dimension** (the total number of data elements in it), or its **size** when all the data is rational (the total number of digits in the data when it is encoded in binary form).

Algorithms are step-by-step procedures for solving problems. We will measure the **computational effort** of an algorithm to solve a problem instance by the number of basic operations (additions, subtractions, multiplications, divisions, comparisons, lookups, etc.) that it takes when it is applied to solve that instance.

We are interested in finding the most efficient (or the fastest, in the sense of requiring the least computational effort in general) algorithms to solve problems. We would expect the relative difficulty of problem instances to increase in general with their largeness measure, so the computational effort of an algorithm should be expressed in terms of this largeness measure. But, even among problem instances with the same largeness measure, the computational effort of an algorithm may vary considerably depending on the actual values for the input data; this makes it very difficult to develop efficiency measures for algorithms.

One possible measure of the efficiency of an algorithm, called the

average computational complexity or the **average time complexity** of the algorithm, is the average computational effort (i.e., the mathematical expectation of the computational effort) expressed as a function of the largeness of the instance, under the assumption that all the input data are random variables with known distributions. Deriving this average complexity requires complete knowledge of the probability distributions of the data in the class of problem instances on which the algorithm will be applied, this kind of detailed information is not available in general. Therefore, we will not discuss this measure.

Another measure of efficiency of an algorithm is the **empirical average computational complexity**(i.e., the observed average computational effort of the algorithm in computational experiments with randomly generated data). This measure is used in an informal way; it is not theoretically satisfactory, since the observed performance of the algorithm may depend critically on the probability distributions used to generate the data in the experiments.

A third measure of the efficiency of an algorithm, known as the **worst case computational complexity**, is a mathematical upper-bound (i.e., the maximum) for the computational effort needed by the algorithm, expressed as a function of the largeness measure of the instance, as the values of the input data elements take all possible values in their range. This is the measure used commonly in computer science to classify algorithms as good or bad; this is the measure that we will use. So, in the text, **computational complexity** of an algorithm refers to this worst case computational complexity function.

When n is some measure of how large a problem instance is (either the size, or the dimension), an algorithm for solving it is said to be of order $n^{\mathbf{r}}$ or $\mathrm{O}(n^{\mathbf{r}})$ if its worst case computational effort grows as $\alpha n^{\mathbf{r}}$, where α and r are numbers that are independent of the largeness measure and the data in the instance. The n and m that appear in our computational complexity measures are usually the number of nodes and lines in the network.

As an example, consider the following problem that came up in the previous discussion: Given a connected undirected network $\mathrm{G} = (\mathcal{N}, \mathcal{A})$ with $|\mathcal{N}| = n \geq 2$, $|\mathcal{A}| = m$, check whether all its nodes have even degree. We now state a simple algorithm for this problem formally

in the style that we will use and determine its computational complexity. The algorithm scans each edge once. It maintains numbers called degree indices for the nodes, which become the degrees at termination. The list is the set of unscanned edges.

Step 1 Initialization For each $i \in \mathcal{N}$, set d_i, its degree index, to 0. List $= \mathcal{A}$.

Step 2 Select an Edge to Scan If list $= \emptyset$, go to Step 4. Otherwise, select an edge e from the list to scan. Delete e from the list.

Step 3 Scanning Let e be the edge to be scanned. Add 1 to the degree indices of each of the two nodes on e. Go to Step 2.

Step 4 Termination The vector of degree indices, $d = (d_i : i \in \mathcal{N})$ at this stage is the vector of degrees of the nodes in G. Check whether all the d_i in it are even and find the answer to the problem. Terminate.

The work in this algorithm consists of $2m$ additions, and checking the evenness of n integers (i.e., a total of $2m + n$ operations, in terms of these operations). It will be shown later that $m \geq n - 1$ in this problem, so the computational effort of this algorithm in terms of these operations is $\leq 4m$; hence the computational complexity of this algorithm is $O(m)$. Therefore, given a connected undirected network with m edges, the existence of an Euler route in it can be checked with an effort of $O(m)$, by the previous discussion.

In this simple problem n, m are parameters describing the dimension of the problem, there is no numerical data, and the computational effort of the algorithm for it is $2m + n$. In the algorithms that follow, there will be numerical data, and the computational effort in them usually depends not just on the dimension or size of the problem but on the actual values of these numbers and possibly on the manner in which the algorithm is executed (specific rules used to select edges or nodes to scan, which may have been left open in the statement of the algorithm, etc.). Therefore, to determine the computational complexity of these algorithms it will be necessary to determine the maximum

possible effort (or a reasonably close upper bound for it) as these data elements take all possible values in their range.

An algorithm whose time complexity function is bounded above by a polynomial function in the size of the problem instance is called a **polynomial time algorithm**. If it is not a polynomial time algorithm, it may have **exponential time complexity** or some other complexity in the worst case.

A Network Model

We will now provide a network model for a typical production distribution problem.

A company makes chairs at 3 plants with wood, for which there are 2 suppliers, and sells them through 3 wholesalers. Relevant data is given in the following tables.

The whole process here can be seen as a flow of wood from the suppliers, through the plants where it is converted into chairs, to the wholesalers. There are limits, both lower and upper, on the amounts of flow through the various points, and either costs or revenues associated with these flows.

On an average a chair requires 20 lbs. of wood. It is convenient to define a *chair unit* of wood to be 20 lbs. and measure all flows in terms of chairs/day. The revenue obtained by selling chairs is treated as negative cost.

Nodes in the network are S_1, S_2 (representing the two suppliers for wood), P_1, P_2, P_3 (the three plants), and W_1, W_2, W_3 (the wholesalers). Arcs in the network represent channels along which material is shipped

Plant	Production cost \$/*chair* at plant	Production capacity in chairs/day	Lower bound on chairs made/day
1	8	1200	0
2	4	600	200
3	5	450	300

Supplier	Cost of shipping wood ($/lb) to plant			Minimum quantity to be purchased from supplier	Selling price of wood ($/lb)
	1	2	3		
1	.02	.03	.04	8 tons	.10
2	.05	.03	.03	10 tons	.075

Plant	Cost of shipping ($/chair) to wholesaler		
	1	2	3
1	2	1	1
2	1.5	2	1
3	1	1.5	2
Selling price ($/chair)	25	20	22
Max. chairs wanted/day	2100	1600	1700
Min. chairs wanted/day	500	400	300

between pairs of nodes. The flow amount on an arc (x, y) represents the amount of material (chair units/day) shipped from x to y.

Figure 1.6 shows the network model. The 3 numbers on a node are the lower and upper bounds for total flow either originating at that node (for S_1, S_2) or passing through that node (for P_1, P_2, P_3), or being shipped to that node (W_1, W_2, W_3), and the unit cost associated with that flow. The number on each arc is the cost/unit flow on it. Remember that a ton of wood is 100 chair units. The problem is to find a flow vector in this network that minimizes the total cost subject to the bounds specified.

In real world applications, the networks usually involve very large numbers of nodes and arcs, but the mathematical structure of the problem remains very similar to that in this example.

Neglecting parallel lines, the total number of lines in a directed network G $= (\mathcal{N}, \mathcal{A})$ with $|\mathcal{N}| = n$, $|\mathcal{A}| = m$, can be at most $n(n-1)$.

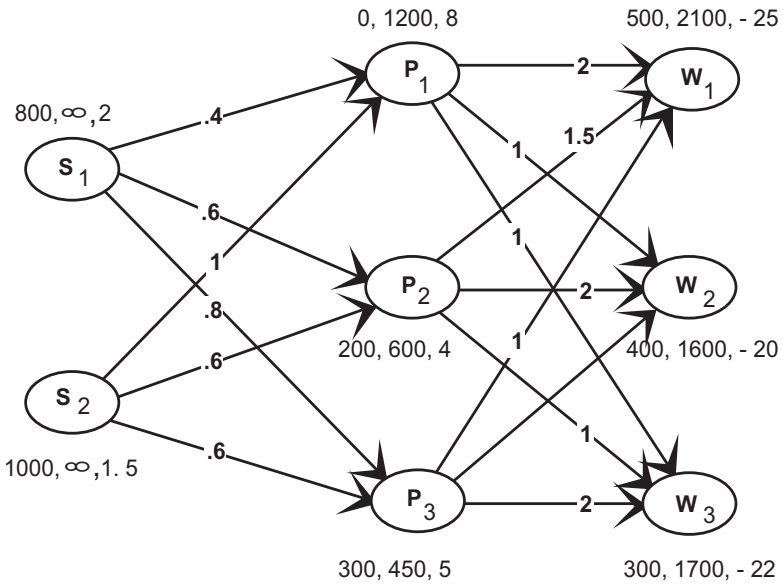


Figure 1.6: Network for the chairmaker's problem. Lower and upper bounds for flow on each arc are 0, ∞, respectively. Other data explained in para above.

If m is close to this number, the network is said to be **dense**. If m is much less than $n(n-1)$, the network is said to be **sparse**. The **sparsity** of G is measured by how small m is in comparison with $n(n-1)$. In very sparse networks, the number of nonparallel lines incident at any node will be much smaller than the possible $n-1$. The networks occurring in most practical applications tend to be very sparse.

1.2 Notation and Preliminaries

In this section, G will denote the network $(\mathcal{N}, \mathcal{A})$.

Let **X, Y** be two subsets of nodes, not necessarily disjoint. If G is undirected, the symbol (**X; Y**) or (**Y;X**) denotes the set of all edges in G with one node in **X** and another node in **Y**. For example, in the network in Figure 1.5, ({1, 2} ; {3, 4}) = $\{e_3, e_2, e_5, e_6\}$ and ({1, 2, 3}

; {2, 4}) = $\{e_1, e_4, e_5, e_6, e_7\}$.

If G is directed, the symbol (**X, Y**) denotes the set of arcs in G with tail in **X** and head in **Y**. So, in the directed case (**X, Y**), (**Y, X**) may not be the same. For example, in the network in Figure 1.6, ({ P_1, S_1, S_2}, { S_1, P_3, W_2}) = $\{(P_1, W_2), (S_1, P_3), (S_2, P_3)\}$.

1.2.1 Linear Programming Background

The **standard form** for an **LP** (we will use this abbreviation for **Linear Program**) is (1.1), where A is a given real matrix of order $m \times n$ and $c = (c_j) \in R^n$. Without any loss of generality, we assume that the rank of A is m; otherwise, if (1.1) is feasible, some of the equality constraints in it must be redundant, and they can be eliminated one at a time until in the remaining system the matrix of coefficients is of full row rank. We also assume that each column vector of A is nonzero. We denote by $A_{i.}, A_{.j}$ the i-th row vector, j-th column vector of A.

$$\begin{array}{lrcl} \text{Minimize} & z(x) = cx & & \\ \text{Subject to} & Ax & = & b \\ & x & \geqq & 0 \end{array} \tag{1.1}$$

In (1.1) $A_{.j}$ is said to be the **original column**, and c_j the **original cost coefficient** of $x_j, j = 1$ to n; $b = (b_i)$ is called **the vector of original right hand side constants** in (1.1).

A **solution** for (1.1) is a vector x satisfying $Ax = b$, whether or not it satisfies $x \geqq 0$. A **feasible solution** is an x satisfying both $Ax = b$ and $x \geqq 0$.

A solution x for (1.1) is said to be a **basic solution** if the set $\{A_{.j}$: j such that $x_j \neq 0\}$ is linearly independent. Assuming that A is of full row rank, a **basis** for (1.1) is a square nonsingular submatrix of A of order m. If $B = (A_{.j_1}, \ldots, A_{.j_m})$ is a basis for (1.1), the vector of variables associated with the columns in B, namely $x_B = (x_{j_1}, \ldots, x_{j_m})$, is known as the **basic vector**, and x_D, the vector of all the variables not in x_B is known as the **nonbasic vector**, associated with it. Let D denote the submatrix of A consisting of the columns of nonbasic

variables. Then, after rearranging the variables, the constraints in (1.1) can be written as:

$$\begin{aligned} Bx_B + Dx_D &= b \\ x_B \geqq 0, x_D &\geqq 0 \end{aligned} \tag{1.2}$$

Then the **basic solution** of (1.1) corresponding to this basic vector is obtained by setting $x_D = 0$ in (1.2) (i.e., fixing all the nonbasic variables at their lower bound, which is the only finite bound on them) and then solving (1.2) for the values of x_B. This is given by:

$$x_D = 0, x_B = B^{-1}b \tag{1.3}$$

The basic solution is said to be **degenerate** if at least one of the components in $B^{-1}b$ is zero, **nondegenerate** otherwise. Thus every basis for (1.1) leads to a basic solution; and conversely, every basic solution of (1.1) corresponds to at least one basis for (1.1).

The basis B and the basic vector x_B for (1.1) are said to be **primal feasible** if the basic solution in (1.3) is feasible to (1.1) (i.e., if $B^{-1}b \geqq 0$); otherwise they are **primal infeasible**. In the former case, the solution in (1.3) is said to be a **basic feasible solution** abbreviated as **BFS** for (1.1). Each BFS corresponds to an $\mathbb{R}^n$ extreme point of the set of feasible solutions for (1.1), and vice versa. See Murty [1983].

Canonical Tableaus

The **canonical tableau** of (1.1) wrt the basis B or the basic vector x_B is obtained by multiplying the system of equality constraints in it on the left by B^{-1}. It is given below.

Basic variables	x	
x_B	$B^{-1}A$	$B^{-1}b$

When the basic and nonbasic columns are rearranged in proper order as in (1.2), the canonical tableau becomes:

Basic variables	x_B	x_D	
x_B	I	$B^{-1}D$	$B^{-1}b = \bar{b}$

The vector $\bar{b}$ is known as the **updated right hand side constants vector** in the canonical tableau. The column of x_j in the canonical tableau, $B^{-1}A_{.j} = \bar{A}_{.j}$ is called **the updated column of** x_j *wrt the basis B, or the associated basic vector* x_B. This can be computed from the original column of this variable, and the basis inverse.

Interpretation of the Updated Column of a Nonbasic Variable as the Representation of Its Original Column

The updated column $\bar{A}_{.j} = (\bar{a}_{1j}, \ldots, \bar{a}_{mj})^T$ of x_j in the canonical tableau of (1. 1) wrt the basis $B = (A_{.1}, \ldots, A_{.m})$ say, where $A_{.1}, \ldots, A_{.m}$ are the original columns of the basic variables in x_B, is $B^{-1}A_{.j}$. So, $A_{.j} = B\bar{A}_{.j} = \bar{a}_{1j}A_{.1} + \ldots + \bar{a}_{mj}A_{.m}$ (i.e., *the updated column* $\bar{A}_{.j}$ *is the vector of coefficients in the representation of the original column* $A_{.j}$ *as a linear combination of the basic columns*, see Murty[1983]). This is an important result relating updated columns in canonical tableaus for (1.1) with the corresponding original columns and the columns of the basis. This result is used later in deriving the canonical tableau for the system of flow conservation equations of a single commodity flow problem combinatorially without performing any pivot steps.

EXAMPLE 1.1

Consider the following system of constraints for an LP.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	b
1	-1	0	1	1	0	0	1
0	-3	1	1	1	1	0	2
1	0	1	0	1	1	1	7

$x_j \geq 0$ for all j

Consider the basic vector $x_{B_1} = (x_1, x_3, x_4)$ for this problem. The corresponding basis is B_1.

$$B_1 = (A_{.1}, A_{.3}, A_{.4}) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \; B_1^{-1} = \begin{pmatrix} 1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{pmatrix}$$

Canonical Tableau wrt The Basic Vector (x_1, x_3, x_4)

Basic variables	x_1	x_2	x_3	x_4	x_5	x_6	x_7	b
x_1	1	1	0	0	1/2	0	1/2	3
x_3	0	-1	1	0	1/2	1	1/2	4
x_4	0	-2	0	1	1/2	0	$-1/2$	-2

We denote by $A_{.j}$ the original column of x_j, and by $\bar{A}_{.j}$ its updated column in the above canonical tableau. So, $\bar{A}_{.2} = (1, -1, -2)^T$ and it can be verified that $A_{.2} = A_{.1} - A_{.3} - 2A_{.4}$, as discussed above.

The basic solution of this system wrt x_{B_1} is $\bar{x} = (3, 0, 4, -2, 0, 0, 0)^T$, and since $x_4 = -2 < 0$ in this solution, it is not a feasible basic solution for this system.

The Dual Problem

Associate the dual variable π_i with the ith equality constraint in (1.1), $i = 1$ to m. Let $\pi = (\pi_1, \ldots, \pi_m)$ be the row vector of dual variables. The dual of (1.1) is:

$$\begin{aligned} &\text{Maximize} && \pi b \\ &\text{Subject to} && \pi A_{.j} \leq c_j, j = 1 \text{ to } n. \end{aligned} \tag{1.4}$$

In matrix notation the constraints in (1.4) are $\pi A \leqq c$. Given the basic vector $x_B = (x_{j_1}, \ldots, x_{j_m})$ and the associated basis B, the row vector $c_B = (c_{j_1}, \ldots, c_{j_m})$ is the associated **original basic cost vector**. The **dual basic solution** corresponding to the basis B is obtained by solving the system of dual constraints corresponding to the basic variables in x_B as equations, i.e.,

$$\pi A_{.j} = c_j, \text{ for each } j = j_1, \ldots, j_m \tag{1.5}$$

or equivalently, $\pi B = c_B$ yielding $\pi = c_B B^{-1}$. The *dual slack vector* corresponding to the basis B is $\bar{c} = (c - \pi A) = (c - c_B B^{-1} A)$, and it is also known as the vector of **reduced *or* relative cost coefficients** in (1.1) wrt the basis B. The dual basic solution obtained from (1.5) is said to be **dual feasible** if it satisfies all the dual constraints in (1.4) (i.e., if $\bar{c} \geqq 0$). If this is satisfied, the basis B is said to be *dual feasible* for (1.1). The BFS corresponding to a primal and dual feasible basis is an *optimum feasible solution* for (1.1).

A feasible solution $\bar{x}$ for (1.1) is optimal iff there exists a dual vector $\bar{\pi}$, such that $\bar{x}, \bar{\pi}$ together satisfy all the following conditions.

$$\begin{array}{rrcl} \text{Primal feasibility} & A\bar{x} & = & b, \bar{x} \geqq 0 \\ \text{Dual feasibility} & \bar{\pi}A & \leqq & c \\ \text{Complementary slackness} & \bar{x}_j(c_j - \bar{\pi}A_{.j}) & = & 0, \text{ for all } j = 1 \text{ to } n \end{array} \tag{1.6}$$

Bounded Variable LPs

A **bounded variable LP** is a problem of the following form:

$$\begin{array}{rrcl} \text{Minimize} & cx & & \\ \text{Subject to} & Ax & = & b \\ & \ell_j \leqq x_j & \leqq & k_j, \text{ for each } j \end{array} \tag{1.7}$$

where A is a matrix of order $m \times n$, and $\ell = (\ell_1, \ldots, \ell_n)^T, k = (k_1, \ldots, k_n)^T$ are given lower bound and upper bound vectors satisfying $\ell \leqq k$. Some

of the k_j may be $+\infty$, but we assume that ℓ is finite. As before, we assume that the rank of A is m, and that each column of A contains at least one nonzero entry. Let $B = (A_{.j_1}, \ldots, A_{.j_m})$ be a nonsingular square submatrix of A of order m, and $x_B = (x_{j_1}, \ldots, x_{j_m})$. Then x_B is the *basic vector* corresponding to the basis B for (1.7). Basic solutions of (1.7) correspond to **partitions of the variables**, (x_B, x_L, x_U), where x_B is a basic vector and x_L, x_U are the vectors of nonbasic variables made equal to their lower, upper bounds respectively in the basic solution. Notice that x_U should be such, that for every x_j in it k_j must be finite. The basic solution corresponding to this partition is obtained by solving for the values of the basic variables in x_B from the system of equations in (1.7) after fixing the nonbasic variables in x_L, x_U at their respective bounds. It is:

$$\begin{aligned} x_j &= \ell_j \text{ if } x_j \text{ is in } x_L, \text{ or } k_j \text{ if } x_j \text{ is in } x_U \\ x_B &= B^{-1}\Big(b - \sum(\ell_j A_{.j} : \text{ over } j \text{ s. t. } x_j \in x_L) \\ &\qquad - \sum(k_j A_{.j} : \text{ over } j \text{ s. t. } x_j \in x_U)\Big) \end{aligned} \tag{1.8}$$

The basic solution in (1.8) is said to be **degenerate** if the value of at least one basic variable is equal to either its lower or upper bound; **nondegenerate** otherwise. The solution in (1.8) is feasible to (1.7) if the value of every basic variable satisfies the bounds on it, and in this case the partition (x_B, x_L, x_U) is said to be a **primal feasible partition**, and the solution itself is called a **basic feasible solution (BFS)**. Thus every BFS for (1.7) is associated with a primal feasible partition for it and vice versa.

Associate a dual variable π_i to the ith equality constraint in (1.7), $i = 1$ to m; dual variables μ_j, γ_j with the bound restrictions on x_j (γ_j is defined only if k_j is finite), $j = 1$ to n. Let $\pi = (\pi_i), \mu = (\mu_j), \gamma = (\gamma_j)$ denote the row vectors of these dual variables. The dual is:

$$\begin{aligned} \text{Maximize} \quad & \pi b + \mu\ell - \sum(\gamma_j k_j \quad : \quad \text{over } j \text{ s. t. } k_j \text{ is finite}) \\ \text{S. to} \quad & \pi A_{.j} + \mu_j - \gamma_j = c_j, \text{ for } j \text{ s. t. } k_j \text{ finite} \\ & \pi A_{.j} + \mu_j = c_j, \text{ for } j \text{ s. t. } k_j \text{ is } \infty \end{aligned} \tag{1.9}$$

$$\mu \geqq 0, \gamma \quad \geqq \quad 0$$

The dual basic solution corresponding to the partition (x_B, x_L, x_U) is defined to be $\pi = c_B B^{-1}$, where c_B is the row vector of original basic cost coefficients, and B is the associated basis. This partition is said to be **dual feasible** if $\bar{c} = c - \pi A = c - c_B B^{-1} A$ satisfies:

$$\bar{c}_j \begin{cases} \geqq 0 & \text{for each } j \text{ such that } x_j \in x_L \\ \leqq 0 & \text{for each } j \text{ such that } x_j \in x_U \end{cases}$$

If these are satisfied, define for each $j = 1$ to n, $\mu_j = \bar{c}_j$ if $\bar{c}_j > 0, \mu_j = 0$ if $\bar{c}_j \leqq 0$; and $\gamma_j = -\bar{c}_j$ if $\bar{c}_j < 0, \gamma_j = 0$ if $\bar{c}_j \geqq 0$. Then verify that $(\pi = c_B B^{-1}, \mu, \gamma)$ satisfy (1.9).

A feasible solution $\bar{x}$ for (1.7) is optimal iff there exists a dual vector $\bar{\pi}$, such that $\bar{x}, \bar{\pi}$ together satisfy **primal feasibility** $(A\bar{x} = b, \ell \leqq \bar{x} \leqq k)$, **dual feasibility** $((c_j - \bar{\pi} A_{.j}) \geqq 0$ for all j such that $k_j = +\infty)$, and the following **complementary slackness conditions for optimality**

$$\begin{array}{rcl} c_j - \bar{\pi} A_{.j} > 0 & \text{implies} & \bar{x}_j = \ell_j \\ c_j - \bar{\pi} A_{.j} < 0 & \text{implies} & \bar{x}_j = k_j \\ \ell_j < \bar{x}_j < k_j & \text{implies} & c_j - \bar{\pi} A_{.j} = 0 \end{array} \tag{1.10}$$

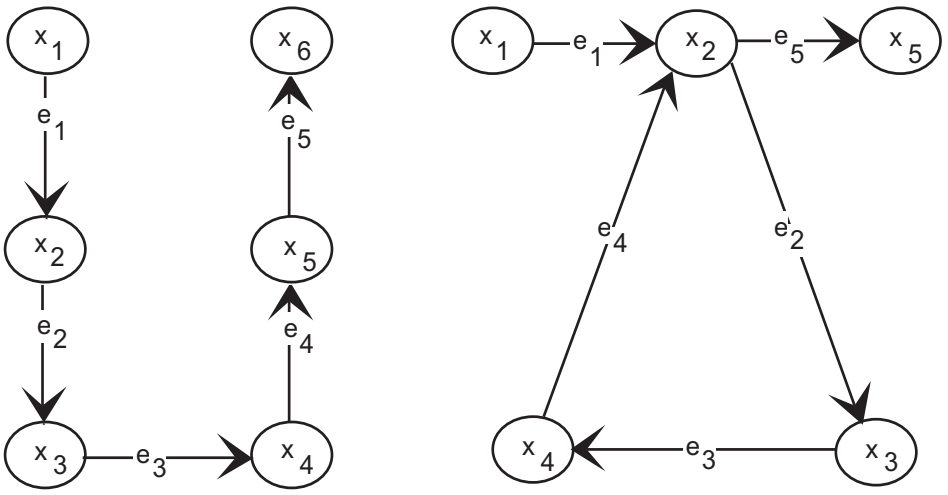
1.2.2 Paths, Chains, Trees, and Other Network Objects

A **path** $\mathcal{P}$ from x_1(**origin or initial node**) to x_k(**destination *or* terminal node**) in G is a sequence of points and lines alternately, $x_1, e_1, x_2, e_2, \ldots, e_{k-1}, x_k$, such that for each $r = 1$ to $k-1$, e_r is either the arc (x_r, x_{r+1}) or the arc (x_{r+1}, x_r) or the edge $(x_r; x_{r+1})$ with some orientation selected for it, which we will again treat as an arc. It is a sequence of lines connecting the points x_1 and x_k, but the lines need not all be directed towards x_k. An arc whose orientation coincides with (is opposite to) the direction of travel from origin to destination is called a **forward (reverse) arc** of the path. A **chain** is a path in which all the arcs are forward arcs. A path (chain) is said to be a

simple path (simple chain) if no point or line is repeated on it. See Figures 1.7 (a), (b), 1.8. A path (chain) is said to be an **elementary path (elementary chain)** if it does not pass through any line more than once, but it may pass through nodes more than once. The chain in Figure 1.7 (b) going through arcs e_1 to e_5 in that order is an elementary, but not simple chain.

The network (**N, A**) where **N** is the set of distinct points and **A** is the set of distinct lines on a path $\mathcal{P}$, is called the **underlying partial subnetwork of *G* corresponding to** $\mathcal{P}$. A **cycle (circuit)** is a path (chain) from a node back to itself satisfying the property that in the underlying partial subnetwork (**N, A**) corresponding to it, each node in **N** is incident to an even number of lines in **A**. A **simple cycle (simple circuit)** is a cycle (circuit) in which no node or line is repeated, except of course for the initial and terminal nodes which are the same. So, a simple cycle is a cycle that does not contain another cycle as a proper subsequence. Also, every cycle must either be simple itself or contain a simple cycle as a subsequence. And every point has degree 2 in the underlying partial subnetwork of a simple cycle. A cycle that does not pass through any line more than once but may pass through nodes more than once is called an **elementary cycle**.

Let $x_1, e_1, x_2, e_2, \ldots, e_{u-1}, x_u$ be a simple path $\mathcal{P}$ from x_1 to $x_u \neq x_1$ in G. We can store $\mathcal{P}$ using **node labels**. The origin x_1 has no **predecessors**, so its **predecessor index (*or* predecessor label)** is $\emptyset$. For $2 \leq r \leq u, x_{r-1}$ is the immediate predecessor of x_r on $\mathcal{P}$, so the predecessor index of x_r is x_{r-1}. To remember the orientation of the arc incident into it, we define the label on x_r to be $(x_{r-1}, +)$ $[(x_{r-1}, -)]$ if e_{r-1} is (x_{r-1}, x_r) $[(x_r, x_{r-1})]$ i.e., a forward [reverse] arc on $\mathcal{P}$. See Figure 1.8. The simple path itself can be traced by a **backwards trace of these predecessor labels** beginning at the terminal node. The last arc on $\mathcal{P}$ is the forward arc (x_{u-1}, x_u) if the label on x_u is $(x_{u-1}, +)$, or the reverse arc (x_u, x_{u-1}) if that label is $(x_{u-1}, -)$. Now go back to the predecessor node x_{u-1}, look up the label on it and continue in the same manner. The trace stops when the node with the $\emptyset$ label, the origin, is reached. In simple chains all the arcs are forward, so for storing them, only the predecessor indices are used without the $+, -$ signs. In all network algorithms, labels of this type are used; that's why they are



(a) A simple chain.

(b) An elementary , but not a simple chain.

Figure 1.7:

called **labeling algorithms**. This labeling scheme is only useful for representing simple paths; verify that it is not adequate to represent nonsimple paths.

A simple cycle in a directed network G can be oriented in one of two possible ways. An **oriented cycle** in a directed network is a simple cycle for which an orientation has been selected. Arcs on an oriented cycle are classified into forward arcs (those whose orientation coincides with that of the cycle), and reverse arcs (those whose orientation is opposite to that of the cycle). Changing the orientation of an oriented cycle interchanges its sets of forward and reverse arcs. As an example, consider the simple cycle 2, (2, 6), 6, (6, 8), 8, (7, 8), 7, (3, 7), 3, (2, 3), 2 in the network in Figure 1.12 (see later on). As it is written, this simple cycle is oriented in the clockwise direction, with {(2, 6), (6, 8)} as forward arcs, and {(2, 3), (3, 7), (7, 8)} as reverse arcs. If this simple cycle is oriented in the anticlockwise direction, these sets switch their roles.

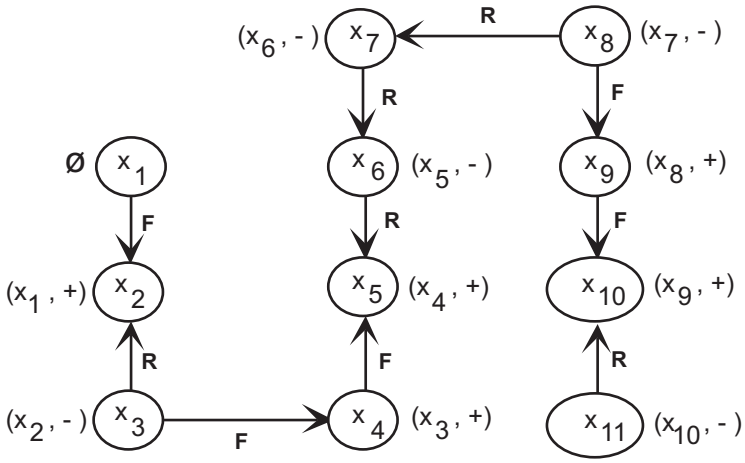


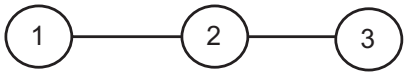
Figure 1.8: A path from x_1 to x_{11}. "F" indicates a forward arc, and "R" indicates a reverse arc. This path is simple. Node labels for storing it are entered by the side of the nodes.

The network G is said to be **connected** if there exists a path between every pair of points in it. A network that is not connected is just two or more independent connected networks put together. The connected networks in it are called its **connected components**.

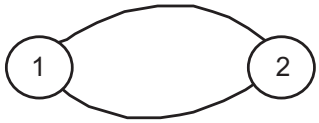
A directed network $G = (\mathcal{N}, \mathcal{A})$ is said to be **strongly connected** if there exists a chain from each node to every other node in it. If the directed network G is not strongly connected, it can be separated into its **strongly connected components**, where, each strongly connected component of G is a maximal partial network of G that is strongly connected.

Exercise

1.1 Suppose $G = (\mathcal{N}, \mathcal{A})$ is a network with $\mathcal{N} = \{1, \ldots, n\}$, and $\mathcal{A}$ is given as a list, listing the nodes on each line. Develop an efficient algorithm to check whether G is connected, and, if it is not, to identify each connected component in it. Determine its computational complexity. Similarly, if G is a directed network, develop an efficient algorithm to check whether it is strongly connected, and, if it is not, to identify its



1, (1, 2), 2, (2, 3), 3, (3, 2), 2, (2, 1), 1 which uses these two edges in forward and backwards directions is not a cycle.



1, $(1, 2)_1$, 2, $(2, 1)_2$,1 consisting of one of these parallel edges in the forward and the other in the reverse direction is a simple cycle.

Figure 1.9:

strongly connected components.

Lower Bounds, Capacities and Node-Arc Flow Variables

Let G = $(\mathcal{N}, \mathcal{A})$ be a directed connected network. In single commodity flow models on G, the flow amount on arc $(i, j) \in \mathcal{A}$ (the amount of commodity transported from node i to node j along this arc, in units per unit time) is denoted by $f_{i,j}$, or $f(i, j)$, or just f_{ij}, and called the **node-arc flow amount *or* flow variable** corresponding to arc (i, j). In applications there are usually lower and upper bounds specified on f_{ij}; these are the **lower bound**, ℓ_{ij} and **capacity**, k_{ij}, of arc $(i, j) \in \mathcal{A}$. The arcs in $\mathcal{A}$ are arranged in some order, and $f = (f_{ij}), \ell = (\ell_{ij}), k = (k_{ij})$ denote the flow, lower bound, and capacity vectors in which these quantities are ordered in the same order as arcs are in $\mathcal{A}$. Often $\ell = 0$, but in some applications it may be nonzero. We will always have $\ell \leq k$.

If $\mathbf{X}$, $\mathbf{Y}$ are two subsets of $\mathcal{N}$, not necessarily disjoint, we define $f(\mathbf{X}, \mathbf{Y}) = \sum(f_{ij} :$ over$(i, j) \in \mathcal{A}$ with $i \in \mathbf{X}, j \in \mathbf{Y})$, i.e., it is the sum of f_{ij} over arcs (i, j) in the set $(\mathbf{X}, \mathbf{Y})$ defined earlier. The symbols $\ell(\mathbf{X}, \mathbf{Y})$, $k(\mathbf{X}, \mathbf{Y})$ carry similar meanings. When $\mathbf{X}$ is a singleton set containing only one node, i say, we denote $f(\mathbf{X}, \mathbf{Y})$, $\ell(\mathbf{X}, \mathbf{Y})$, $k(\mathbf{X}, \mathbf{Y})$ by $f(i, \mathbf{Y})$, $\ell(i, \mathbf{Y})$, and $k(i, \mathbf{Y})$ respectively.

As an example, for the network in Figure 1.6, $f(\{P_1, S_1, S_2\}, \{S_1, P_3, W_2\}) = f_{P_1W_2} + f_{S_1P_3} + f_{S_2P_3}$. In this notation it can be verified that $f(\mathcal{N}, \mathcal{N})$ is the sum of the flow amounts on all the arcs in G; and $f(i, \mathcal{N}), f(\mathcal{N}, i)$ are the sums of the flow amounts on arcs in the forward star, reverse star of i respectively. And if $\mathbf{X}_1, \mathbf{X}_2$ is a partition of $\mathbf{X}$ (i.e., $\mathbf{X}_1 \cup \mathbf{X}_2 = \mathbf{X}$, $\mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset$), and $\mathbf{Y}_1, \mathbf{Y}_2$ is a partition of $\mathbf{Y}$, then $f(\mathbf{X}, \mathbf{Y}) = f(\mathbf{X}, \mathbf{Y}_1) + f(\mathbf{X}, \mathbf{Y}_2) = f(\mathbf{X}_1, \mathbf{Y}) + f(\mathbf{X}_2, \mathbf{Y}) = f(\mathbf{X}_1, \mathbf{Y}_1) + f(\mathbf{X}_1, \mathbf{Y}_2) + f(\mathbf{X}_2, \mathbf{Y}_1) + f(\mathbf{X}_2, \mathbf{Y}_2)$.

In many single commodity flow models, two special nodes, one called the **source node** (which we denote by $\breve{s}$), and another called the **sink node** (which we denote by $\breve{t}$) are specified, and the commodity is required to be shipped from $\breve{s}$ to $\breve{t}$ in G. All the other points are called **intermediate points *or* transit nodes** in these models.

In some models, a unit cost coefficient c_{ij}, the cost per unit flow on arc (i, j), is given for each arc, $c = (c_{ij})$ is the vector of these cost coefficients.

With all this data, the network itself is denoted by the symbol G $= (\mathcal{N}, \mathcal{A}, \ell, k, c, \breve{s}, \breve{t})$. In some models there may be even more data elements, in others less, then the network is denoted by a corresponding symbol consisting of all the data elements in it.

Now consider the case where the network $G = (\mathcal{N}, \mathcal{A})$ is undirected. The edge $(i; j)$ can be treated as the pair of arcs $(i, j), (j, i)$ as in Figure 1.10 (a). In single commodity flow models, a flow of 10 units in the direction i to j and 6 units in the direction j to i as in Figure 1.10 (b), is equivalent to a net flow of 4 units from i to j as in Figure 1.10 (c) (this argument is not valid if there are two are more distinct commodities and the flows from i to j and j to i are of different commodities). So, in single commodity flow models we can assume that each edge in the network will only be used in one of the two possible directions. We assume that lower bounds for flows along all the edges are 0, and that the capacity restriction applies in the direction in which it is used. Under this assumption, each edge in the network can be replaced by a pair of arcs as in Figure 1.10 (a) with the same data holding for both arcs in the pair. Hence, in the study of single commodity flow problems, we assume without any loss of generality that the network is a directed network.

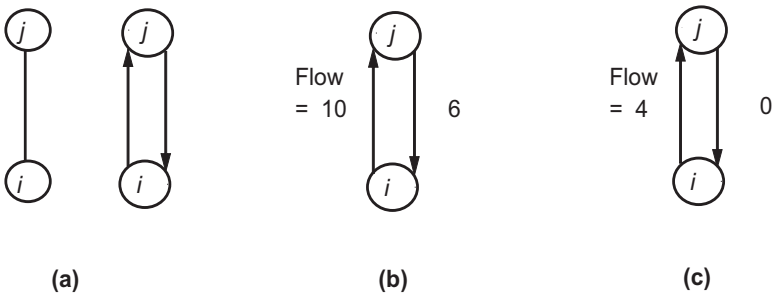


Figure 1.10:

In network models, nodes usually represent processing centers, warehouses etc. In some of these models **node capacities** may be specified. The node capacity of a node represents the maximum amount of material that can either enter or leave the node. It represents the maximum amount of flow that the node can process per unit time. As an illustration, the network model for the chair making company problem in Section 1.1 had node capacities specified. In Section 2.1 (Chapter 2), we show how to transform this model so as to modify node capacities into arc capacities.

Cuts, Cutsets

Let G = $(\mathcal{N}, \mathcal{A})$ be a connected network. A **cut** in G is a subset of lines, the deletion of all of which disconnects the network. First consider the case where G is undirected. Cuts in undirected networks are commonly used in the graph-theoretic study of electrical networks. Let $\mathbf{X} \subset \mathcal{N}, \bar{\mathbf{X}} = \mathcal{N} \backslash \mathbf{X}$ where $\mathbf{X}$, $\bar{\mathbf{X}}$ are both nonempty. This partition of $\mathcal{N}$ generates the cut $(\mathbf{X}; \bar{\mathbf{X}})$ which is the set of all edges with one node in $\mathbf{X}$ and the other in $\bar{\mathbf{X}}$. $(\mathbf{X}; \bar{\mathbf{X}})$ is a **disconnecting set** because it has the **path blocking property** (after deleting all the edges in this set there exists no path in the remaining network from any node in $\mathbf{X}$ to any node in $\bar{\mathbf{X}}$). The **cut vector** of this cut is its 0-1 incidence vector over the set of edges $\mathcal{A}$. For example, in the network in Figure 1.11 the cut ({ 1, 2, 3 }; { 4, 5, 6 }) = { e_3, e_4, e_6, e_7 }, and hence its cut vector is (0,0,1,1,0,1,1,0,0,0) when the edges are arranged in the

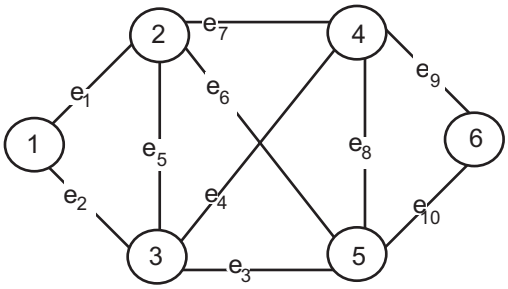


Figure 1.11:

order e_1 to e_{10}. A **cutset** in G is a minimal set of edges whose removal disconnects G (i.e., it is a cut satisfying the property that no proper subset of it is a cut). Equivalently, a cutset in the connected undirected network G, is a minimal set of edges whose removal disconnects G into exactly two connected components. As an example, in Figure 1.11 the cut ({ 1, 2, 3 }; { 4, 5, 6 }) is a cutset. But the cut ({ 1, 6 }; { 2, 3, 4, 5}) is not a cutset.

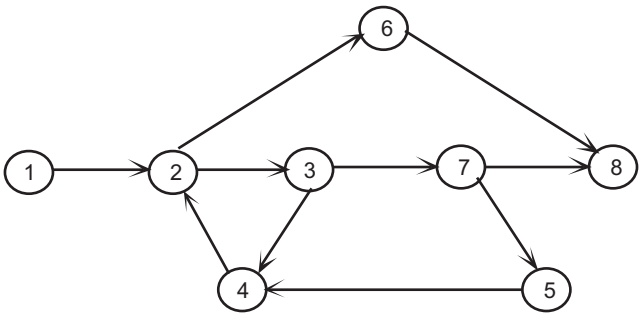


Figure 1.12:

Now consider the case where the connected network G is directed. Let G = $(\mathcal{N}, \mathcal{A}, \ell, k)$. Let $\mathbf{X}$, $\bar{\mathbf{X}}$ be a partition of $\mathcal{N}$ with both the sets nonempty. This partition generates a cut denoted by [$\mathbf{X}$, $\bar{\mathbf{X}}$] it consists of ($\mathbf{X}$, $\bar{\mathbf{X}}$), called the **set of forward arcs of this cut**, and ($\bar{\mathbf{X}}$, $\mathbf{X}$) called the **set of reverse arcs of this cut**. Notice that the order in which the sets $\mathbf{X}$, $\bar{\mathbf{X}}$ are recorded in [$\mathbf{X}$, $\bar{\mathbf{X}}$] is important, switching them exchanges the forward and reverse arc sets in the cut.

In the network remaining after all the forward arcs of the cut [$\mathbf{X}$, $\bar{\mathbf{X}}$] are deleted, there exists no chain from any node in $\mathbf{X}$ to any node in $\bar{\mathbf{X}}$, and if both forward and reverse arcs are deleted there exists no path from any node in $\mathbf{X}$ to any node in $\bar{\mathbf{X}}$. Thus in directed networks while a cut has the path blocking property, the set of forward arcs of a cut by itself has the **chain blocking property**. In network algorithms the concept of a cut plays a role dual to that of a path.

In the single commodity maximum value flow problem on the directed connected network G = $(\mathcal{N}, \mathcal{A}, \ell, k, \breve{s}, \breve{t})$, special cuts called **cuts separating the source $\breve{s}$ and the sink $\breve{t}$** play a significant role. These are cuts [$\mathbf{X}$, $\bar{\mathbf{X}}$] satisfying the property $\breve{s} \in \mathbf{X}$ and $\breve{t} \in \bar{\mathbf{X}}$. So deletion of all the forward arcs of a cut separating $\breve{s}$ and $\breve{t}$ destroys all chains from $\breve{s}$ to $\breve{t}$ even though there may be a path connecting them consisting of reverse arcs, thus no flow is possible from $\breve{s}$ to $\breve{t}$ in the remaining network.

The **capacity** of the cut [$\mathbf{X}$, $\bar{\mathbf{X}}$] in G = $(\mathcal{N}, \mathcal{A}, \ell, k)$ is defined to be $k(\mathbf{X}, \bar{\mathbf{X}}) - \ell(\bar{\mathbf{X}}, \mathbf{X})$.

Here again, a cut [$\mathbf{X}$, $\bar{\mathbf{X}}$] is called a cutset if it is a minimal cut (i.e., if no proper subset of it is a cut). As an example, consider the single commodity flow network in Figure 1.12. The cut [{ 1, 2, 7 }, { 3, 4, 5, 6, 8 }] has (2, 3), (2, 6), (7, 5), (7, 8) as forward arcs, and (3, 7), (4, 2) as reverse arcs. It is not a cutset since the cut [{ 1, 2, 3, 7, 4, 5 }, { 6, 8 }] with no reverse arcs and only (2, 6), (7, 8) as forward arcs is a proper subset of it.

Forests and Trees

A **forest** in a network G is a partial subnetwork that contains no cycles. In Figure 1.1 the dashed partial subnetwork, which is redrawn in Figure 1.3(b), is a forest. A **tree** in G is a connected partial subnetwork that contains no cycles. Each connected component of a forest is a tree. The forest in Figure 1.3(b) contains two trees. A **spanning tree** in a network is a subnetwork that is a tree. Hence a spanning tree in G = $(\mathcal{N}, \mathcal{A})$ is a subnetwork $(\mathcal{N}, \hat{\mathcal{A}})$ which is connected and contains no cycles. In Figure 1.1 the subnetwork consisting of the thick lines is a spanning tree.

A single isolated node by itself constitutes a tree, which we call the **trivial tree**. It is the only tree which has no lines. Unless it is mentioned otherwise, in the sequel the word tree refers to a nontrivial tree.

A node in a tree $\mathbb{T}$ is called a **terminal node, end node, pendant node,** or **leaf node** if its degree in $\mathbb{T}$ is 1 (i.e., if it is incident to exactly one line in $\mathbb{T}$). For example, in the tree consisting of the thick lines in Figure 1.1, 1 is a terminal node, but 2 is not. In the tree in Figure 1.14 nodes 1, 2, 3, 4, 8, 10 are the terminal nodes. The terminal nodes of a tree are also called its **leaves**. An arc or edge in the tree incident at a leaf node is called a **leaf arc** or **leaf edge** of the tree.

Exercises

1.2 Prove that every nontrivial tree has at least two terminal nodes.

1.3 Prove that if a line e_1 is deleted from a tree $\mathbb{T}$, what is left is a forest. Also, if e_1 is a line incident to a terminal node i of $\mathbb{T}$, then what is left after deleting e_1 from $\mathbb{T}$ is another tree and the trivial tree containing only node i.

1.4 Let G be a connected network with n nodes. Prove that every spanning tree in G contains $n-1$ lines (Hint: use induction on n and the results in previous exercises).

1.5 Let G be a network with n nodes, and $\mathbb{T}$ a subnetwork of G containing $n-1$ lines and no cycles. Prove that $\mathbb{T}$ must be a spanning tree in G. (You have to prove that $\mathbb{T}$ is connected. Prove that any subnetwork like $\mathbb{T}$ must have a point whose degree in $\mathbb{T}$ is one. Use induction.)

1.6 Prove that a connected network in which the number of lines is equal to the number of nodes -1 must be a tree.

1.7 If $\{e_1, \ldots, e_r\}$ is a set of r lines containing no cycles in a connected network, prove that there is at least one spanning tree containing all of these lines.

1.8 Prove that there exists a unique simple path between every pair of distinct nodes in a tree.

1.9 Prove that the number of lines is one half of the sum of the degrees of the points in a network.

1.10 Prove that the number of odd degree nodes in a network must be an even number.

1.11 In a network in which no point has a degree greater than 2, prove that the number of nodes of degree one is an even number.

1.12 Prove that every chain from the source to the sink must contain at least one forward arc of any cut separating them.

1.13 For any nonempty proper subset of nodes $\mathbf{X} \subset \mathcal{N}$ in a connected undirected network $G = (\mathcal{N}, \mathcal{A})$, define $\mathcal{A}_{\mathbf{X}}$ to be the set of edges in $\mathcal{A}$ with both their nodes from $\mathbf{X}$. Let $\bar{\mathbf{X}}$ be the complement of $\mathbf{X}$.

(i) Prove that $(\mathbf{X}; \bar{\mathbf{X}})$ is a cutset of G iff $(\mathbf{X}, \mathcal{A}_{\mathbf{X}})$ and $(\bar{\mathbf{X}}, \mathcal{A}_{\bar{\mathbf{X}}})$ are both connected networks.

(ii) If $\mathbf{S}$ is a cutset of G, and $\mathbf{V}_1, \mathbf{V}_2$ are the node sets of the two connected components of $(\mathcal{N}, \mathcal{A} \backslash \mathbf{S})$, then show that $\mathbf{S} = (\mathbf{V}_1; \mathbf{V}_2)$.

(iii) Prove that a cutset of G contains at least one in-tree edge of every spanning tree in G. Further, show that a subset of edges $\mathbf{S}$ is a cutset in G iff it is a minimal set of edges containing at least one in-tree edge of every spanning tree in G.

(iv) Prove that a cycle and a cutset of G have an even number of common edges.

Let $G = (\mathcal{N}, \mathcal{A})$ be a connected network with $|\mathcal{N}| = n, |\mathcal{A}| = m$. A subset of lines $\mathbf{A} \subset \mathcal{A}$ is said to be a **cotree** in G iff its complement $\mathcal{A} \backslash \mathbf{A}$ is the set of lines in a spanning tree for G. The word "cotree" is

an abbreviation for **complement of a spanning tree**. So, for **A** to be a cotree in G, a necessary condition is that $|\mathbf{A}| = m - n + 1$.

When considering a spanning tree $\mathbb{T}$ in a network G, a line in G is said to be an **in-tree arc** or **in-tree edge** if it lies in $\mathbb{T}$; otherwise it is said to be an **out-of-tree arc (edge)**. The set of out-of-tree lines defines the cotree corresponding to $\mathbb{T}$.

Let i, j be the nodes on an out-of-tree line e in a spanning tree $\mathbb{T}$. By Exercise 1.8, there exists a unique simple path $\mathcal{P}$ in $\mathbb{T}$ between i and j. Hence, when e is included in $\mathbb{T}$ a unique simple cycle containing e is created (it consists of e and the path $\mathcal{P}$), this is known as the **fundamental cycle of** e wrt $\mathbb{T}$. As an example, in the spanning tree consisting of the thick lines in Figure 1.1, (6, 5) is an out-of-tree arc. The fundamental cycle associated with it is 6, (6, 5), 5, (4, 5), 4, (4, 2), 2, (2, 7), 7, (6, 7), 6. All the arcs on this cycle are in-tree arcs except (6, 5).

By replacing an in-tree arc in the fundamental cycle associated with the out-of-tree arc (i, j), by (i, j), a new spanning tree is obtained. As we will see later, every basis for a single commodity pure network flow problem in a connected directed network corresponds to a spanning tree. And when this problem is solved by the simplex algorithm, every pivot step is exactly the operation of obtaining a new spanning tree by adding an out-of-tree arc and dropping an in-tree arc in its fundamental cycle.

Trees are stored using node labels. These **tree labels** make it possible to store in the computer all the information necessary to manipulate a tree. Their use has led to enormous improvements in the efficiency of computer implementations of network algorithms. Some of the node labels are: **predecessor index** *(P)*, **successor index** *(S)*, **elder brother index** *(EB)*, **younger brother index** *(YB)*,and **thread label** *(TH)*. Besides these, the distance of the node, the number of successors of the node, and others are sometimes used in network codes. The most commonly used of these labels are defined below. Each label used in the data structure requires an array of length $n = |\mathcal{N}|$, and it imposes the work of updating this label whenever the tree changes in the algorithm.

We will first discuss tree labels for storing a spanning tree $\mathbb{T}$ in a

connected directed network G = $(\mathcal{N}, \mathcal{A})$ with $n = |\mathcal{N}|$. Modifications to be made if G is undirected are mentioned later. Select any node and designate it as the **root node**. Once the root node is selected, the tree is called a **rooted tree**. A rooted tree is a tree with one of its nodes identified as the root. The labels are generated by the following procedure while drawing the tree with the root node at the top and the other nodes below it level by level. In this procedure, nodes may be in three possible states: unlabeled, labeled and unscanned, or labeled and scanned. The present sets of unlabeled, labeled and unscanned nodes are denoted by **Y, X** respectively.

Initialization Make the P, YB, EB indices of the root node all $\emptyset$, and $\mathbf{X} = \{\text{root node}\}$, $\mathbf{Y}$ = set of all non-root nodes.

Step 1 Select a node to be scanned Terminate if $\mathbf{X} = \emptyset$. Otherwise, select a node from **X** to scan.

Step 2 Scanning a node Let i be the node to be scanned, delete it from **X**. Find $\mathbf{J} = \{j : j \in \mathbf{Y}$ and j is joined to i by an in-tree arc $\}$. Nodes in **J** are the **sons** or **children** or **immediate successors** of i, and i is their **parent** or **immediate predecessor**.

If $\mathbf{J} = \emptyset$, i has no children, define its successor index $S(i) = \emptyset$.

If $\mathbf{J} \neq \emptyset$, arrange the nodes in **J** in some order, say $j_1, \ldots, j_r$. Then j_1 is the eldest child of i. j_p is an elder brother of j_q (and j_q is an younger brother of j_p) if $p < q$. The successor index of i, $S(i)$ is defined to be $-j_1[+j_1]$ if the in-tree arc joining i and j_1 is $(i, j_1)[(j_1, i)]$. For each $u = 1$ to r, define the predecessor index of j_u, $P(j_u)$, to be $+i[-i]$ if the in-tree arc joining i and j_u is $(i, j_u)[(j_u, i)]$. For each $u = 1$ to r, define the elder brother index, $EB(j_u)$ to be $\emptyset$ if $u = 1, j_{u-1}$ if $u > 1$ and the younger brother index $YB(j_u)$ to be $\emptyset$ if $u = r, j_{u+1}$ if $u \leq r - 1$. Nodes $j_1, \ldots, j_r$ are now labeled and unscanned; transfer them from **Y** to **X**. Go to Step 1.

Thus, the elder brother index of any node is the youngest among its elder brothers, and its younger brother index is the eldest among its

younger brothers, when these brothers exist. Figure 1.13 explains our convention for the signs of the predecessor and successor indices. They indicate the orientation of the in-tree arc joining a node and its parent.

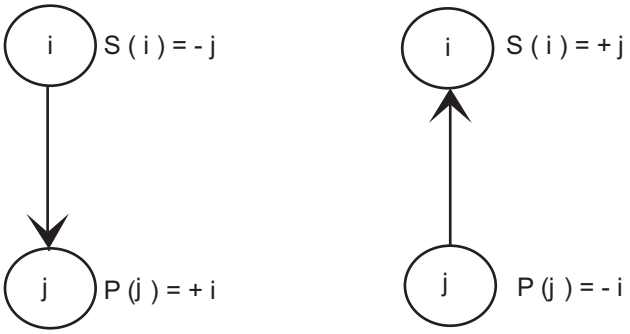


Figure 1.13: Signs of indices when i is parent of j and the in-tree arc joining them is (i, j) (on left), or (j, i) (on right). The successor index of i, S(i) is $\pm j$ only if j is the eldest child of i.

All nonroot terminal nodes have no successor. Conversely, if $\mathrm{S}(i) = \emptyset$, i must be a nonroot terminal node.

The $\pm$ signs on the successor and predecessor indices are used only when dealing with a directed network. For a spanning tree in an undirected network, all the indices are defined in exactly the same way with the exception that the successor and predecessor indices carry no signs.

EXAMPLE 1.2

For the purpose of this illustration, only the in-tree arcs are given in Figure 1.14; all the out-of-tree arcs and the remaining data on the network is omitted. We assume that nodes which are brothers of each other are arranged from left to right in Figure 1.14 for determining the elder, younger brother relationships. This leads to the predecessor, successor, and brother labels in the table given below, for the spanning tree in Figure 1.14.

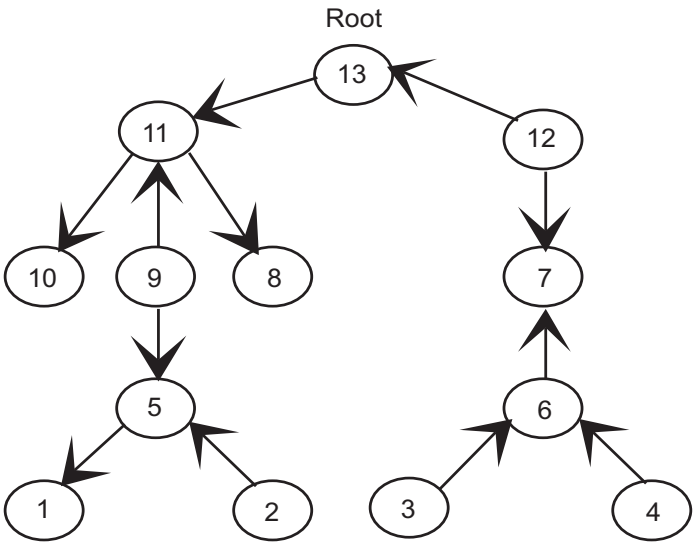


Figure 1.14: A spanning tree

Predecessor, successor, brother labels for tree in Fig. 1.14

Node i	1	2	3	4	5	6	7	8	9	10	11	12	13
P(i)	+5	-5	-6	-6	+9	-7	+12	+11	-11	+11	+13	-13	$\emptyset$
S(i)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	-1	+3	+6	$\emptyset$	-5	$\emptyset$	-10	-7	-11
YB(i)	2	$\emptyset$	4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	8	9	12	$\emptyset$	$\emptyset$
EB(i)	$\emptyset$	1	$\emptyset$	3	$\emptyset$	$\emptyset$	$\emptyset$	9	10	$\emptyset$	$\emptyset$	11	$\emptyset$

13 is the root node

The unique path in a rooted tree $\mathbb{T}$ from a node j to the root node is called the **predecessor path of** j in $\mathbb{T}$. It can be found by a backward trace of the predecessor indices beginning with j recursively. If $\mathrm{P}(j) = +i$ $[-i]$, (i, j) $[(j, i)]$ is the first arc in this path. Now look up the node $\mathrm{P}(i)$ and continue in the same manner until the root node is reached. As an example, the predecessor path of node 1 in the spanning tree in Figure 1.14 is 1, (5, 1), 5, (9, 5), 9, (9, 11), 11, (13, 11), 13.

A node i is said to be an **ancestor** or **predecessor** of another node j in the rooted tree $\mathbb{T}$ if i appears on the predecessor path of j in $\mathbb{T}$, in this case j is a **descendent** or **successor** of i.

The **family** of a node j in the rooted tree $\mathbb{T}$ is the set consisting of j and all the descendents of j, it is denoted by the symbol $\mathbf{H}(\mathbb{T}, j)$.

The **level** of a node in a rooted tree is defined to be the number of lines on its predecessor path. Thus, the root node is the only level 0 node in a rooted tree. For any r, given the set of level r nodes, level $r+1$ is empty if none of the level r nodes have a successor, otherwise level $r+1$ consists of the set all immediate successors of nodes in level r.

Let (i, j) be an in-tree arc in a rooted tree $\mathbb{T}$. Then, one of the nodes among i, j must be a parent and the other its child. If i is the parent and j the child (i.e., $\mathrm{P}(j) = +i$) this arc is said to be **directed away from the root node**. On the other hand, if j is the parent and i the child, this arc is said to be **directed towards the root node**. See Figure 1.15.

For each in-tree line e in a rooted tree $\mathbb{T}$, **son**(e), **parent**(e) refer to the son, parent nodes on it. Thus, if $\mathbb{T}$ is a directed network and the in-tree arc (i, j) is directed away from [towards the] root node, $\mathrm{son}(i, j) = j$, $\mathrm{parent}(i, j) = i$ [$\mathrm{son}(i, j) = i$, $\mathrm{parent}(i, j) = j$].

The set of younger brothers of a node j is empty if $\mathrm{YB}(j) = \emptyset$, or is the union of $\{\mathrm{YB}(j)\}$ and the set of younger brothers of $\mathrm{YB}(j)$ otherwise. Using this recursively, the set of younger brothers of any node can be found efficiently. In a similar manner, the set of elder brothers of any node can be found recursively using only the EB indices. The set of brothers of a node is the union of its sets of younger and elder brothers.

The set of immediate successors of a node j is empty if $\mathrm{S}(j) = \emptyset$, or is the union of $\{\mathrm{S}(j)\}$ and the set of younger brothers of $\mathrm{S}(j)$ otherwise.

The **set of descendents** of a node j is empty if $\mathrm{S}(j) = \emptyset$. Otherwise it is the union of the set of immediate successors of j and the sets of descendents of each of the immediate successors of j. The thread label defined later, is designed to obtain this set very efficiently.

Let i, j be two nonroot nodes in a rooted tree $\mathbb{T}$. The first common node on the predecessor paths of i and j is known as the **apex** on the simple path between i and j in $\mathbb{T}$. The simple path between i and j in $\mathbb{T}$ is obtained by putting the predecessor paths of i and j together and eliminating the common lines on them. As an example, the simple

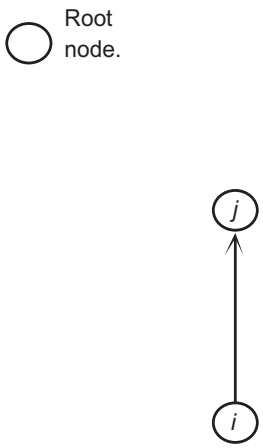


Figure 1.15: If $\text{son}(i,j) = j$ as on the left [$\text{son}(i,j) = i$ as on the right] arc (i,j) is directed away from [towards the] root node.

path between 1 and 10 in the spanning tree in Figure 1.14 is 1, (5, 1), 5, (9, 5), 9, (9, 11), 11, (11, 10), 10. Node 11 is the apex on this path.

If $\mathbb{T}$ is a spanning tree in G, and (i,j) is an out-of-tree arc, the fundamental cycle of (i,j) wrt $\mathbb{T}$ consists of arc (i,j) and the simple path in $\mathbb{T}$ from j to i. As an example, the fundamental cycle of arc (10, 1) (not in the figure) wrt the spanning tree in Figure 1.14 is 10, (10, 1), 1, (5, 1), 5, (9, 5), 9, (9, 11), 11, (11, 10), 10.

Let $\mathbb{T}$ be a rooted spanning tree in a connected network $\mathrm{G} = (\mathcal{N}, \mathcal{A})$ with $|\mathcal{N}| = n$. Let e be an in-tree line. If G is undirected let $\bar{\mathbf{X}} = \mathbf{H}(\mathbb{T}, \text{son}(e))$. If G is directed let $\bar{\mathbf{X}}$ be the set among $\mathbf{H}(\mathbb{T}, \text{son}(e))$ and its complement, which contains $\text{head}(e)$. Let $\mathbf{X} = \mathcal{N} \backslash \bar{\mathbf{X}}$. Let $\mathrm{G}_{\mathbf{X}}, \mathrm{G}_{\bar{\mathbf{X}}}$ denote the partial networks of G, and $\mathbb{T}_{\mathbf{X}}, \mathbb{T}_{\bar{\mathbf{X}}}$ the partial networks of $\mathbb{T}$, induced by the sets of nodes $\mathbf{X}$,$\bar{\mathbf{X}}$. $\mathbb{T}_{\mathbf{X}}, \mathbb{T}_{\bar{\mathbf{X}}}$ are themselves spanning trees in $\mathrm{G}_{\mathbf{X}}, \mathrm{G}_{\bar{\mathbf{X}}}$; one or both of them may be trivial trees. So, both $\mathrm{G}_{\mathbf{X}}, \mathrm{G}_{\bar{\mathbf{X}}}$ are connected networks.$\mathbf{X}$, $\bar{\mathbf{X}}$ is a partition of the node set in G, this partition generates a cut in G. If G is undirected and ($\mathbf{X}$; $\bar{\mathbf{X}}$) is

the cut, it is a cutset called the **fundamental cutset corresponding to the in-tree edge**e in $\mathbb{T}$. If G is directed, the cut is [$\mathbf{X}$, $\bar{\mathbf{X}}$]; it is also a cutset and is called the fundamental cutset corresponding to the in-tree arc e in $\mathbb{T}$. The only in-tree line in this fundamental cutset is e. Each in-tree line leads to a different fundamental cutset; together they form the **set of fundamental cutsets** wrt $\mathbb{T}$.

Thus, given a spanning tree $\mathbb{T}$ in a network G, each line in $\mathbb{T}$ defines a fundamental cutset in G, and each line in the corresponding cotree defines a fundamental cycle.

We now present some of the other tree labels used in network codes. One is the **number of successors** denoted by NS(i) for node i. Another is the **distance** or **depth label** which is the level of the node in the tree.

Another commonly used label is the **thread label**. Let $\mathcal{N} = \{1, \ldots, n\}$ be the set of nodes in a rooted tree $\mathbb{T}$. A thread label for $\mathbb{T}$ is a one-to-one correspondence from $\mathcal{N}$ onto $\mathcal{N}$ satisfying certain properties. Given such a correspondence $t_i, i \in \mathcal{N}$, define other maps $t_i^r, i \in \mathcal{N}$ by recursion: $t^1(i) = t_i$, for each $i \in \mathcal{N}$; $t^r(i) = t^{r-1}(t^1(i))$, for each $i \in \mathcal{N}, r \geq 2$. Then the correspondence $t_i, i \in \mathcal{N}$ is said to be a **thread label** for $\mathbb{T}$ if for each i such that NS(i) $\neq 0$ the set $\{t^r(i) : r = 1, \ldots,$ NS(i) $\}$ is the set of all descendents of i. Many types of thread labels satisfying these conditions can be defined, but the most commonly used one is defined by:

$$t_i = \begin{cases} \text{eldest son, i.e., S}(i), \text{ if S}(i) \neq \emptyset \\ \\ \text{YB}(i), \text{ if S}(i) = \emptyset \text{ and YB}(i) \neq \emptyset \\ \\ \text{YB index of first ancestor with a YB, if } i \text{ has no son or YB} \\ \\ \text{root node, otherwise} \end{cases}$$

Given the thread labels (t_i), the set of all descendents of node i is the largest set of the form $\{t_i, t^2(i), \ldots, t^k(i)\}$ such that the parent of $t^k(i)$ is one of the nodes $i, t_i, \ldots, t^{k-1}(i)$. So, we can find the set of all descendents of any node i efficiently by recursive application of the maps $t^r(i)$ for all r up to NS(i). As this is a commonly used

operation in network algorithms, maintaining the thread label can be very advantageous. Also, the result in Exercise 1.19 states that the number of $\emptyset$ entries in S(.) and YB(.) indices together is $n+1$. This indicates that the information in S(.) and YB(.) can be stored more economically in the single thread index.

Given the thread label $t_i, i \in \mathcal{N}$, the **preorder distance label** , PD(i), and **the last successor label** , LS(i), corresponding to it, can be defined for each $i \in \mathcal{N}$ as below.

$$\text{PD}(i) = \begin{cases} 1, \text{ if } i \text{ is the root} \\ r+1, \text{ if } i \neq \text{root, where } r \text{ is s.t. } i = t^r(root) \end{cases}$$

$$\text{LS}(i) = \begin{cases} i, \text{ if } i \text{ has no successor} \\ \\ t^r(i) \text{ otherwise, for } r \text{ s.t. } t^r(i) \text{ is a descendent of } i, \\ \text{but } t^{r+1}(i) \text{ is not.} \end{cases}$$

For the rooted tree in Figure 1.14, we provide these labels in the following table. Figure 1.16 illustrates the thread labels for a rooted tree with pointers.

NS(i), distance (d_i), thread index (t_i), PD(i), and LS(i) node labels for the rooted tree in Figure 1.18

Node i	1	2	3	4	5	6	7	8	9	10	11	12	13
NS(i)	0	0	0	0	2	2	3	0	3	0	6	4	12
d_i	4	4	4	4	3	3	2	2	2	2	1	1	0
t_i	2	8	4	13	1	3	6	12	5	9	10	7	11
PD(i)	6	7	12	13	5	11	10	8	4	3	2	9	1
LS(i)	1	2	3	4	2	4	4	8	2	10	8	4	4

Thread labels facilitate the forward traversal of the tree, an operation performed many times in simplex based network codes. It can be viewed as a connecting link or thread which passes through each node exactly once in a top to bottom, left to right sequence starting from the root node.

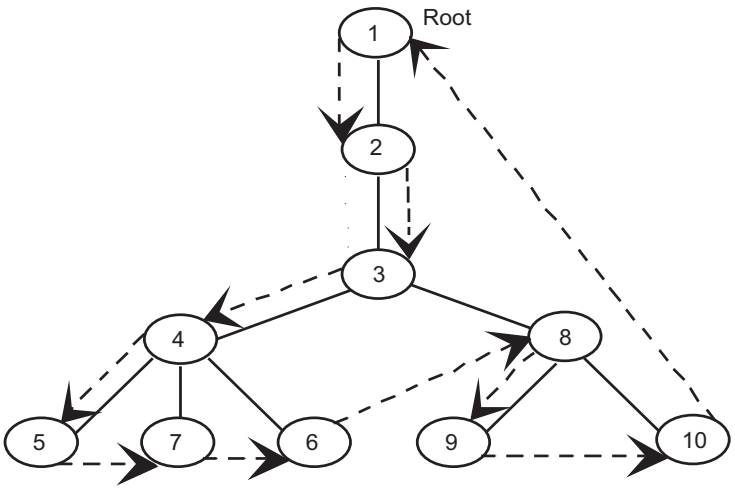


Figure 1.16: The thread labels. The tree consists of solid lines. The dotted arc (i, j) indicates that the thread label of i is j.

These tree labels are mainly used in simplex based algorithms for network flow problems discussed in Chapter 5. Each basic vector for the problem corresponds to a spanning tree, and a pivot step in the algorithm consists of changing the spanning tree by adding an out-of-tree arc to replace an in-tree arc in its fundamental cycle. The tree changes by an arc in each step, and the tree labels are updated by very efficient updating schemes. While the predecessor indices are enough to trace the predecessor paths, the other indices make it possible for updating the tree labels and the node price vector efficiently. Also, the updating of the thread label can be carried out very efficiently. That's why many codes for the network simplex method use the thread label.

An **outtree** or a **branching** is a directed network which is a rooted tree such that every arc on it is directed away from the root node. It has exactly one arc incident into every node other than the root, which has no arcs incident into it. See Figure 1.17. Similarly, an **intree** or **arborescence** is a directed network which is a rooted tree such that every arc in it is directed towards the root node (i.e., the predecessor path of every node is a chain from that node to the root). It has exactly

one arc incident out of every node other than the root, which has no arcs incident out of it.

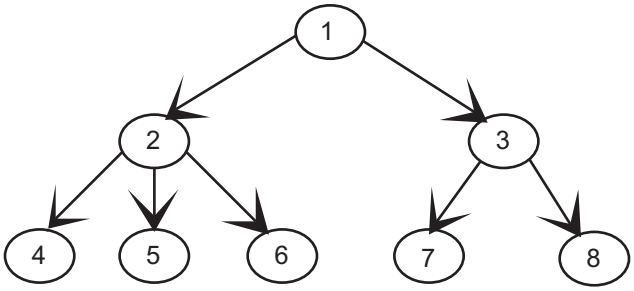


Figure 1.17: An outtree or branching with node 1 as the root.

Tree Growth Subroutines

Several of the algorithms for maximum value flow problems (Chapter 2), shortest chain problems (Chapter 4), minimum cost spanning tree problems in undirected networks (Chapter 9) and blossom algorithms for matching/edge covering problems (Chapter 10) use a scheme that begins by selecting a node in the network, say p, and labeling it with $\emptyset$. At this initial stage, p is the only labeled node, all the other nodes are unlabeled.

General step A labeled node, say i, and one of its adjacent unlabeled nodes, say j, are selected by some rule (this rule is problem dependent, the line joining i, j has to satisfy certain properties for the pair i, j to be selected) and j is labeled with i as its predecessor index. The line joining i, j is known as the **line used in labeling node j**. Now the scheme either terminates or moves on to the next step.

At any stage of this scheme, let $\mathbf{X}$ denote the set of labeled nodes and let $\mathcal{A}(\mathbf{X})$ denote the set of all lines used so far in labeling the nodes in $\mathbf{X}$. We have the following theorem.

THEOREM 1.1 *At every stage of this scheme, the partial subnetwork ($\mathbf{X}$, $\mathcal{A}(\mathbf{X})$) will be a tree spanning the nodes in the set $\mathbf{X}$. It is a rooted tree with its root at node p.*

Proof Initially, the partial subnetwork ($\mathbf{X}$, $\mathcal{A}(\mathbf{X})$) is ($\{ p \}$, $\emptyset$), the trivial tree consisting of node p. In each step, one new node is added to the set $\mathbf{X}$, if it is j, its immediate predecessor is an adjacent node which is already in $\mathbf{X}$, and the line joining it to j is added as a new arc to the set $\mathcal{A}(\mathbf{X})$. This implies that ($\mathbf{X}$, $\mathcal{A}(\mathbf{X})$) is always connected and that $|\mathcal{A}(\mathbf{X})| = |\mathbf{X}| - 1$. So, by the result in Exercise 1.6, ($\mathbf{X}$, $\mathcal{A}(\mathbf{X})$) is always a tree spanning the nodes in $\mathbf{X}$. Also, every node in $\mathbf{X}$ has a unique immediate predecessor, except the node p which has no predecessor, so, these predecessor labels make ($\mathbf{X}$, $\mathcal{A}(\mathbf{X})$) a rooted tree with node p as the root node. ∎

So, this scheme is actually a **tree growth subroutine**, growing a rooted tree with its root node at p. Each step of the scheme is known as a **tree growth step**. It adds one new node and a line connecting it to an earlier in-tree node, to the tree. At any stage of this scheme, the in-tree nodes are the labelled nodes, and the in-tree lines are the lines joining each in-tree node and its immediate predecessor. In these schemes, sometimes several trees may be grown simultaneously in the network. The predecessor indices serve all the functions needed in these schemes, and no successor or brother indices are maintained.

Methods for Selecting a Spanning Tree in a Network

Here we discuss two algorithms for selecting a spanning tree in a connected network G = $(\mathcal{N}, \mathcal{A})$, to initiate algorithms such as the primal network simplex method discussed in Section 5.5.

ALGORITHM 1 : Initialization Here nodes may be in 3 possible states: unlabeled, labeled and unscanned, or labeled and scanned. **List** always refers to the set of labeled and unscanned nodes. Select the root node, say n, and label it with $\emptyset$, and put it in the list. All the other nodes are unlabeled initially.

General Step In a general step, select a node, say i, from the list to scan. Delete i from the list. Find all unlabeled nodes j such that either (i, j), or (j, i), or edge $(i; j)$ is in $\mathcal{A}$. For each such node j include one of the lines joining it to i as an in-tree line. All these nodes are children of i, give predecessor, EB, YB indices to them

and include them in the list, and give the successor index to i as discussed above. If there are no unlabeled nodes left, we have a spanning tree, terminate. If there are some unlabeled nodes but the list is empty, G is not connected, terminate. Otherwise go to the next stage.

In this algorithm, if nodes in the list are maintained in the order in which they are labeled and in each stage the node for scanning is selected by the FIFO rule (First In First Out, or first labeled first scanned rule), then it is called the **breadth-first search method**, and the spanning tree generated a **breadth-first search spanning tree** in G.

ALGORITHM 2 : Initialization This algorithm introduces one node and line into the tree per step. $\mathbf{A}$ denotes the set of lines in $\mathcal{A}$ joining an in-tree and an out-of-tree node at the present stage. This set is maintained in the algorithm. If at some stage there are out-of-tree nodes, but $\mathbf{A} = \emptyset$, G is not connected and the algorithm terminates. Select the root node, say n and declare it as an in-tree node. Make $\mathbf{A}$ = set of all lines incident at n.

General Step In a general step, select a line from $\mathbf{A}$. Introduce this line and the unlabeled node on it, say j, into the tree. If there are no out-of-tree nodes, we have a spanning tree, terminate. Otherwise, delete from $\mathbf{A}$ all lines incident at j that are currently in it. Include in $\mathbf{A}$ all lines joining j to an out-of-tree node. Go to the next step.

In this algorithm, if lines in the set $\mathbf{A}$ are maintained in the order in which they are introduced into this set, and in each step the line selected from it is chosen by the LIFO rule (Last In First Out rule, i.e., the line chosen is always the one put into $\mathbf{A}$ most recently), then the algorithm is called the **depth-first search method, or DFS**, and the spanning tree generated a **depth-first search, or DFS spanning tree** in G. A numbering of the nodes in the order of becoming in-tree nodes is called a **DFS numbering**.

Exercises

1.14 Let $\mathbb{T}_1$ denote a breadth-first search spanning tree in G with root n. For each node $i \neq n$, prove that the predecessor path of i in $\mathbb{T}_1$ is a shortest path between i and n (i.e., it contains the smallest number of lines). Also prove that there cannot be an out-of-tree arc wrt $\mathbb{T}_1$ which joins a node and one of its descendents.

1.15 Let $\mathbb{T}_2$ be a DFS spanning tree in G with root node n. Prove that every line in G connects two nodes, one of which is an ancestor of the other. Also prove that if i, j are leaf nodes in $\mathbb{T}_2$, then there is no line joining i and j in G.

1.16 Prove that every connected network has at least one spanning tree.

1.17 Let G be a connected network. Prove that every cotree $\mathbf{A}$ in G is a cutset-free subset of $\mathcal{A}$. And for any cycle-free subset $\mathbf{B} \subset \mathcal{A}$, prove that there exists a spanning tree in G including it. For any cutset-free subset $\mathbf{A} \subset \mathcal{A}$, prove that there exists a cotree in G including it.

1.18 Let $G = (\mathcal{N}, \mathcal{A})$ be a connected undirected network and $\mathbb{T}$ a spanning tree in it.

(i) If e is an out-of-tree edge, prove that the fundamental cycle of e consists of exactly those in-tree edges of $\mathbb{T}$ whose fundamental cutsets contain e.

(ii) Prove that the fundamental cutset of the in-tree edge e_1 consists of exactly those out-of-tree edges whose fundamental cycles contain e_1.

1.19 Let $\mathbb{T}$ be a rooted tree with n nodes. Let n_1, n_2 be the number of nodes in it for which the S(.), YB(.) indices are respectively $\emptyset$. Prove that the number of nodes for which the EB(.) is $\emptyset$ is n_2. Also prove that $n_1 + n_2 = n + 1$.

Bipartite Networks

A simple cycle is said to be an **odd cycle (even cycle)** if it contains an odd (even) number of lines.

A network G = $(\mathcal{N}, \mathcal{A})$ is said to be a **bipartite network** if $\mathcal{N}$ can be partitioned into two nonempty subsets $\mathcal{N}_1$ and $\mathcal{N}_2$ such that every line in $\mathcal{A}$ joins a point in $\mathcal{N}_1$ with a point in $\mathcal{N}_2$ (i.e., there are no lines in $\mathcal{A}$ joining a pair of points both of which are either in $\mathcal{N}_1$ or in $\mathcal{N}_2$). The partition $(\mathcal{N}_1, \mathcal{N}_2)$ is then called a **bipartition** of G, and G itself denoted by $(\mathcal{N}_1, \mathcal{N}_2 ; \mathcal{A})$. See Figure 1.18. As it is drawn, the bipartiteness of the network in Figure 1.6 may not be apparent, but taking the partition $\mathcal{N}_1 = \{S_1, S_2, W_1, W_2, W_3\}$, $\mathcal{N}_2 = \{P_1, P_2, P_3\}$ it can easily be verified to be so.

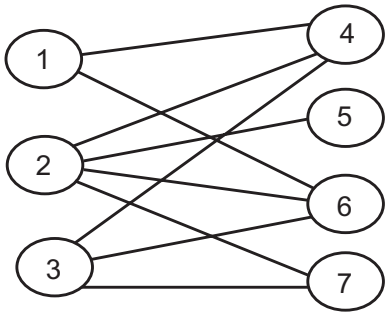


Figure 1.18: A bipartite network. Here ({1,2,3}, {4,5,6,7}) is the bipartition.

THEOREM 1.2 *A network G = $(\mathcal{N}, \mathcal{A})$ is bipartite iff it contains no odd cycles.*

Proof Clearly a network is bipartite iff each of its connected components is. Without any loss of generality we assume that G is connected, because otherwise the proof can be repeated for each connected component separately.

Suppose G is bipartite, and $(\mathcal{N}_1, \mathcal{N}_2)$ is a bipartition for it. While traversing any cycle in G we move alternately to points in the sets $\mathcal{N}_1$, $\mathcal{N}_2$, and hence every cycle in G must be an even cycle.

Now suppose G is a network which contains no odd cycles. Let 1 be an arbitrary point in $\mathcal{N}$. Define $\mathcal{N}_1 = \{i :$ either $i = 1$, or there exists

a simple path from 1 to i with an even number of lines}, $\mathcal{N}_2 = \mathcal{N} \backslash \mathcal{N}_1$. Suppose two points $j, p \in \mathcal{N}_1$ are joined by a line $(j, p) \in \mathcal{A}$. By definition of $\mathcal{N}_1$, there are simple paths $\mathcal{P}_1$ between 1 and j, and $\mathcal{P}_2$ between 1 and p, both with an even number of lines. Let r = number of common lines on $\mathcal{P}_1$, $\mathcal{P}_2$; r_1 = number of lines on $\mathcal{P}_1$ not on $\mathcal{P}_2$; and r_2 = number of lines on $\mathcal{P}_2$ not on $\mathcal{P}_1$. Since $r + r_1, r + r_2$ are both even, $r_1 + r_2$ is also even and it is > 0 since $j \neq p$. So, combining (j, p) with the lines on $\mathcal{P}_1$, $\mathcal{P}_2$ and then eliminating all the common lines on $\mathcal{P}_1$ and $\mathcal{P}_2$ leaves a cycle with $r_1 + r_2 + 1$ = odd number of lines. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ are simple paths, this cycle decomposes into a collection of line-disjoint simple cycles, at least one of which must be an odd cycle, contradicting the hypothesis.

Suppose two points j, p in $\mathcal{N}_2$ are joined by a line $(j, p) \in \mathcal{A}$. Since G is connected, there are simple paths in G between 1 and j, or p, and by the definition of $\mathcal{N}_1$ all these paths must traverse through an odd number of lines. Either there exists a simple path between 1 and j not containing p, or there exists a simple path between 1 and p not containing j; suppose the first possibility holds. Take any simple path from 1 to j not passing through p, and add the line (j, p) at its end. This leads to a simple path from 1 to p traversing through an even number of lines, contradicting $p \in \mathcal{N}_2$. Hence there cannot be any line in G joining a pair of points in $\mathcal{N}_2$. Hence $(\mathcal{N}_1, \mathcal{N}_2)$ is a bipartition for G, and it is bipartite. ∎

The following algorithm can be used to check whether a network $G = (\mathcal{N}, \mathcal{A})$ is bipartite and generate a bipartition for it, if it is. We assume that G is connected, otherwise apply the algorithm on each connected component of G and put the results together. **X** denotes the set of included but unscanned nodes, and **Y** denotes the set of unincluded nodes at any stage.

Step 1 Initialization Select a node, say $1 \in \mathcal{N}$. Make $\mathcal{N}_1 = \{ 1 \}$, $\mathcal{N}_2 = \emptyset$, $\mathbf{X} = \{ 1 \}$, $\mathbf{Y} = \mathcal{N} \backslash \{ 1 \}$.

Step 2 Select a node to scan If $\mathbf{X} = \emptyset$, go to Step 4. Otherwise, select a node from **X** to scan.

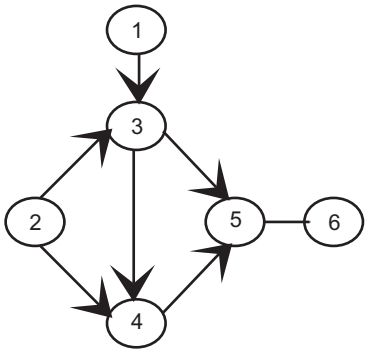
Step 3 Scanning Let i be the node to be scanned, delete it from $\mathbf{X}$. Let $\mathcal{N}'$ denote the set among the pair $\mathcal{N}_1$, $\mathcal{N}_2$ containing node i, and $\mathcal{N}''$ denote the other set in this pair. Let $\mathbf{J}$ be the set of all nodes in $\mathbf{Y}$ which are adjacent to i. If $\mathbf{J} \neq \emptyset$, check whether there is a line in $\mathcal{A}$ joining a node in $\mathbf{J}$ with a node in $\mathcal{N}''$, if so go to Step 5. Otherwise, delete all nodes in $\mathbf{J}$ from $\mathbf{Y}$ and include them in both $\mathbf{X}$ and $\mathcal{N}''$, and return to Step 2.

Step 4 Termination The subsets $\mathcal{N}_1$, $\mathcal{N}_2$ at this stage form a bipartition for G. Terminate.

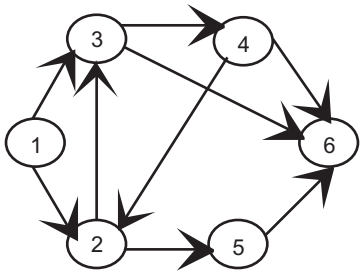
Step 5 Proof of non-bipartiteness G is not bipartite, because two nodes which should belong to the same set in the partition are adjacent. Terminate.

Acyclic Networks

A directed network is said to be an **acyclic network** if it contains no circuits. It may contain cycles, but not circuits. See Figure 1.19.



An acyclic network with acyclic numbering of nodes.



A non-acyclic network. It has the circuit with arcs (2, 3), (3, 4), (4, 2).

Figure 1.19:

THEOREM 1.3 *A directed network $G = (\mathcal{N}, \mathcal{A})$ is acyclic iff its nodes can be numbered so that the number of tail(e) < the number of head(e) for each $e \in \mathcal{A}$.*

Proof If G is acyclic, there must be at least one point i in it whose before set $\mathbf{B}(i) = \emptyset$ (otherwise, by tracing the arcs backwards one can construct a circuit). Suppose there are r_1 points like this, number them 1, 2, ..., r_1 in any order, and delete them and all the arcs incident at them from G. Search the resulting network for points whose before set in it is empty, and number them starting with $r_1 + 1$. Repetition of this process leads to the desired numbering.

If the nodes in G are numbered so that $i < j$ for each $(i, j) \in \mathcal{A}$, clearly there cannot be any circuit in G; hence it is acyclic. ∎

A numbering of the nodes in a directed network G in serial order so that the number of tail(e) is < the number of head(e) for all arcs e is called an **acyclic numbering** or **topological ordering**. The acyclic numbering of the nodes in the network on the left in Figure 1.19 is obtained by applying the procedure in the proof of Theorem 1.3.

Incidence Matrices

Let G be a directed network $(\mathcal{N}, \mathcal{A})$ with n points and m arcs, and no self-loops. Let E be the $n \times m$ matrix that has a row associated with each point in G and a column associated with each arc in G, where the column associated with $(i, j) \in \mathcal{A}$ has only two nonzero entries, a "1" entry in the row associated with point i and a "−1" entry in the row associated with point j. E, defined only for directed networks with no self-loops, is known as the **node-arc incidence matrix** of G. As an example, the node-arc incidence matrix for the network in Figure 1.20 is given below.

Arc →	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9
Node 1	1	0	0	0	0	0	−1	1	1
2	−1	−1	0	0	0	1	0	0	0
3	0	1	1	0	−1	0	1	0	0
4	0	0	0	1	1	−1	0	0	−1
5	0	0	−1	−1	0	0	0	−1	0

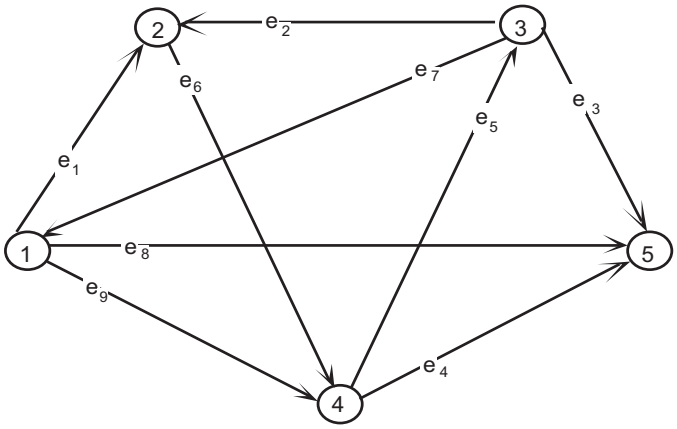


Figure 1.20:

Verify that the set of row vectors of a node-arc incidence matrix is linearly dependent since their sum is 0. Also verify that the sum of node-arc incidence vectors of arcs in a circuit is 0. Verify this for the circuit in Figure 1.20 consisting of arcs e_9, e_5, e_7. The general property is stated in the following Exercise 1.22.

Exercises

1.20 Check whether the network in Figure 1.21 is bipartite.

1.21 Check whether the network in Figure 1.22 is acyclic. If so, provide an acyclic numbering of its nodes.

1.22 Let $e_1, \ldots, e_r$ be the sequence of arcs in a cycle in a directed network G with node-arc incidence matrix E. Orient this cycle some way. Multiply the column vector of E associated with e_t by $+1$ if e_t is a forward arc, or by -1 if it is a reverse arc of the cycle, and add over $t = 1$ to r, show that this gives 0. Thus show that the set of column vectors of E associated with arcs in a set containing a cycle form a linearly dependent set for which there is a linear dependence relation with all the coefficients 0, or ± 1.

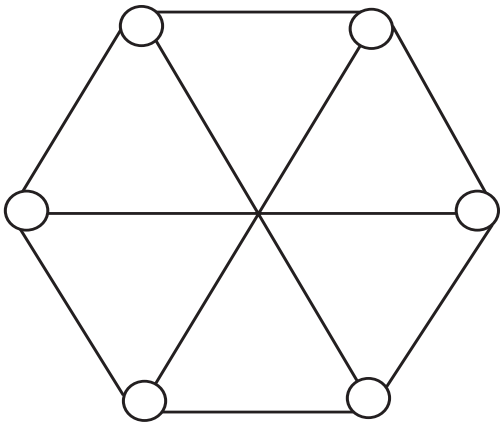


Figure 1.21:

1.23 Prove that the set of columns of the node-arc incidence matrix E of a directed network, associated with the arcs in a set that contains no cycles, must be linearly independent. (This set forms a forest. So, there must be at least one terminal node. Use this repeatedly.)

1.24 Prove that the set of columns in E associated with arcs in a spanning tree is linearly independent. (Use the fact that a tree has at least one terminal node repeatedly.)

1.25 If E is the node-arc incidence matrix of a connected directed network with n points, prove that its rank is $n-1$. Also, prove that any one of the row vectors of E could be deleted as a dependent row, and the remaining matrix will be of full row rank.

1.26 If the directed network G has n points and consists of p connected components, prove that its node-arc incidence matrix has rank $n-p$.

A nonsingular square matrix $D = (d_{ij})$ is said to be an **upper triangular matrix** if $d_{ij} = 0$ for all $i > j$. It is **lower triangular** if D^T is upper triangular, i.e., if $d_{ij} = 0$ for all $j > i$. A **triangular**

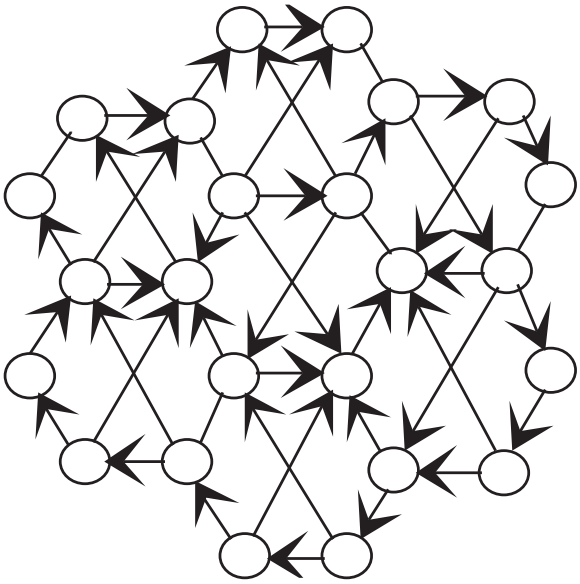


Figure 1.22:

matrix is a nonsingular square matrix that becomes a lower triangular matrix after a permutation of its columns and/or rows. A square matrix is triangular iff it satisfies the following properties.

1. The matrix has a row (or column) that contains a single nonzero entry.

2. The submatrix obtained from the matrix by striking off the row (or column) containing a single nonzero entry and the column (row) in which that entry lies, also satisfies Property 1. The same process can be repeated until all the rows and columns in the matrix are struck off.

For example, the following matrix A can be verified to be triangular, while B is not.

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

If D is a triangular matrix, the system of equations $Dy = d$ can be solved efficiently by back substitution. Identify the equation containing a single nonzero entry on the left hand side; solve that equation for the value of the variable associated with the nonzero coefficient in that equation; substitute the value of this variable in all the remaining equations and continue in the same manner with the remaining system. The same back substitution method can be applied to solve the system of equations $\pi D = c$ when D is triangular.

THEOREM 1.4 *Every nonsingular square submatrix of E, the node-arc incidence matrix of a directed network* G*, is triangular.*

Proof Let D be a nonsingular square submatrix of order r of E. Since every column vector of E contains only two nonzero entries, a $+1$ and a -1, the total number of nonzero entries in D is at most $2r$. If it is $2r$, each column of D contains a $+1$ and a -1, and the sum of all the rows of D is 0, contradicting its nonsingularity. So the total number of nonzero entries in D is at most $2r - 1$. Since D is nonsingular each row of D must contain at least one nonzero entry. As D has r rows, these facts imply that there must be at least one row of D with a single nonzero entry, and that entry is either $+1$ or -1. The same argument applies to the submatrix of D obtained by striking off a row containing a nonzero entry, and the column of that entry. So D satisfies properties 1 and 2 mentioned above for triangularity, and hence is triangular. ∎

COROLLARY 1.1 *Let $\bar{E}$ be the matrix obtained by deleting any row vector from E, the node-arc incidence matrix of a connected directed network* G *with n nodes and m arcs. $\bar{E}$ of order $(n-1) \times m$ is of full row rank. Let B be a basis for $\bar{E}$, and $d = (d_1, \ldots, d_{n-1})^T, c =$*

$(c_1, \ldots, c_{n-1})$. *Consider the system of equations,* $By = d$. *Since* B *is triangular, and all the entries in it are 0, or* ± 1, *when this system is solved by the back substitution method discussed above, its solution will be of the form* $y = (y_j)$ *with each* $y_j = \sum_{i=1}^{n-1} \alpha_i d_i$, *where all the* α_i *are 0, or* ± 1. *similarly, the solution to the system of equations,* $\pi B = c$, *is of the form,* $\pi = (\pi_i)$ *with each* $\pi_i = \sum \beta_j c_j$, *where all the* β_j *are 0, or* ± 1. *Hence, if* d, c *are integer vectors, the solutions to these systems are also integer vectors.*

A real matrix $A = (a_{ij})$ is said to be **totally unimodular** if the determinant of every square submatrix of it is either 0, or ± 1. Since each element a_{ij} can be looked at as the entry in a square submatrix of A of order 1, if A is totally unimodular, all a_{ij} have to be 0, or ± 1. As examples, consider the following matrices:

$$A = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

The determinant of A is -1, and clearly A is totally unimodular. The determinant of B is -2, and so B is not totally unimodular.

THEOREM 1.5 *TOTAL UNIMODULARITY PROPERTY The node-arc incidence matrix* E *of a directed network is totally unimodular.*

Proof We have shown in Theorem 1.4 that every nonsingular square submatrix of E is triangular. This, combined with the fact that all the entries in it are 0 or ± 1, implies that the determinant of every nonsingular square submatrix of E is ± 1, proving the theorem. ∎

We will now discuss a result relating total unimodularity to integrality of extreme points of polyhedra, due to Hoffman and Kruskal [1958].

THEOREM 1.6 *Let* A *be an integer matrix of order* $m \times n$, *and let* $\mathbf{K}(b)$ *be the set of feasible solutions of*

$$\begin{aligned} Ax &\leq b \\ x &\geq 0 \end{aligned}$$

The following three conditions are equivalent.

(i) *A is totally unimodular.*

(ii) *For all integral b such that $\mathbf{K}(b) \neq \emptyset$, all the extreme points of $\mathbf{K}(b)$ are integer.*

(iii) *Every nonsingular square submatrix of A has an integer inverse.*

Proof The fact that (i) implies (ii) follows by applying Cramer's rule. The fact that (i) implies (iii) follows from the definitions.

Introduce slack variables and write down the system of constraints defining $\mathbf{K}(b)$ as

$$\begin{aligned} Ax + Is &= b \\ x, s &\geq 0 \end{aligned} \tag{1.11}$$

Let D be a square submatrix of A of order r which is nonsingular. We will now show that (ii) implies that D^{-1} is an integer matrix. After rearranging the variables and the constraints if necessary, we can assume that D is the submatrix of A contained in rows 1 to r and columns 1 to r. Let B denote the basis for (1.11) consisting of columns $A_{\cdot 1}, \ldots, A_{\cdot r}$ and $I_{\cdot r+1}, \ldots, I_{\cdot m}$, and let the associated basic vector be denoted by y. So, B has the form given below, where I_{m-r} is the unit matrix of order $m - r$, and hence B^{-1} has the form given.

$$B = \left(\begin{array}{c|c} D & 0 \\ \hline F & I_{m-r} \end{array}\right), B^{-1} = \left(\begin{array}{c|c} D^{-1} & 0 \\ \hline -FD^{-1} & I_{m-r} \end{array}\right)$$

Let I denote the unit matrix of order m, and let i be an integer between 1 to r. Select an integer column vector $\xi \in \mathbb{R}^m$ such that $\xi + B^{-1}I_{\cdot i} \geq 0$. Take b to be $B(\xi + B^{-1}I_{\cdot i}) = B\xi + I_{\cdot i}$, which is an integer vector since $B\xi, I_{\cdot i}$ are integer vectors. With this b vector, the BFS of (1.11) corresponding to the basis B is

$$\text{all nonbasic variables } = 0,\ y = \xi + B^{-1}I_{\cdot i} \geq 0 \tag{1.12}$$

Since this b vector is integer, by (ii) the y in (1.12) must be integer, i.e., since ξ is an integer vector, $B^{-1}I_{\cdot i}$ must be integer, and hence

$(D^{-1})_{.i}$ must be integer. Since this is true for all $i = 1$ to r, D^{-1} must be an integer matrix. That is, (ii) implies (iii).

We will now show that (iii) implies (i). Let D be any square nonsingular submatrix of A. Since D^{-1} is integer by (iii), the determinant of (D^{-1}) is integer, and since this is 1/determinant(D), determinant(D) must be ± 1. Since this applies to all square nonsingular submatrices of A, A must be totally unimodular. So, (iii) implies (i).

Hence (i), (ii), (iii) are equivalent. ∎

As a generalization of total unimodularity, an integer matrix A of order $m \times n$ and rank r is said to be **unimodular** if every one of its square submatrices of order r has determinant 0, or ± 1. There is no condition on the determinants of A of order $r - 1$ or less. Clearly every totally unimodular matrix is unimodular, but the converse may not be true. As an example, the matrix given below is unimodular but not totally. This is followed by a theorem relating unimodularity to integrality of extreme points of a polyhedron defined by constraints different from those in the previous theorem.

$$\begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 2 & 1 & 0 & -2 & -1 & 0 \\ 2 & 2 & 1 & -2 & -2 & -1 \end{pmatrix}$$

THEOREM 1.7 *Let A be a given integer matrix of order $m \times n$. Consider the system*

$$Ax = b, x \geq 0 \tag{1.13}$$

Without any loss of generality we assume that rank(A) is m. The following statements are equivalent.

(i) *A is unimodular*

(ii) *Whenever b is an integer vector, every BFS of (1.13) is an integer vector.*

(iii) *Every basis for (1.13) has an integer inverse.*

Proof The proof of this theorem is very similar to that of the previous. We will first show that (ii) implies (iii). Let B be a basis for (1.13) associated with the basic vector x_B. Select an i between 1 to m, and choose an integer column vector $\xi \in \mathbb{R}^m$ such that $\xi + B^{-1}I_{.i} \geq 0$. Take b in (1.13) to be $B(\xi + B^{-1}I_{.i}) = B\xi + I_{.i}$, which is integer. With this integer b-vector, the BFS of (1.13) corresponding to the basis B is $x_B = B^{-1}b = \xi + B^{-1}I_{.i} \geq 0$, and all nonbasic variables = 0. Since ξ is an integer vector, by (ii), $B^{-1}I_{.i} = (B^{-1})_{.i}$ must be an integer vector. Since this is true for all i, B^{-1} must be an integer matrix. So, (ii) implies (iii).

To show that (iii) implies (i), let B be a basis for (1.13). Since determinant$(B^{-1}) = 1/($determinant$(B))$, if B^{-1} is an integer matrix, determinant(B^{-1}) is integer, and hence determinant(B) is ± 1. Since this must hold for all bases for (1.13), (i) must hold.

The fact that (i) implies (ii) follows by using Cramer's rule. Hence (i), (ii), (iii) are equivalent. ∎

Let G $= (\mathcal{N}, \mathcal{A})$ be a connected directed network with n nodes, and m arcs, and let $\mathbb{T}$ be a spanning tree in G. Let $e_1, \ldots, e_{n-1}$ be the arcs in $\mathbb{T}$, and $e_n, e_{n+1}, \ldots, e_m$ be the out-of-tree arcs. Orient each fundamental cycle so that the out-of-tree arc on it is a forward arc. For $p = n$ to m and $t = 1$ to m define

$$\lambda_{tp} = \begin{cases} 0 & \text{if } e_t \text{ not on fundamental cycle of } e_p \\ +1 & \text{if } e_t \text{ is a reverse arc on this cycle} \\ -1 & \text{if } e_t \text{ is a forward arc on this cycle} \end{cases}$$

For each $p = n$ to m, the row vector $(\lambda_{1p}, \ldots, \lambda_{mp})$ is the **incidence vector of the fundamental cycle of the out-of-tree arc** e_p, and the matrix L of order $(m - n + 1) \times m$ consisting of these row vectors is known as the **fundamental cycle-arc incidence matrix** of G wrt $\mathbb{T}$. And the matrix $\lambda = (\lambda_{tp} : t = 1 \text{ to } n-1, p = n \text{ to } m)$ of order $(n-1) \times (m-n+1)$ is known as the **in-tree arc-fundamental cycle incidence matrix** of G wrt $\mathbb{T}$. The matrix λ is the transpose of the submatrix of L consisting of its columns corresponding to in-tree arcs.

As an example, for the network in Figure 1.20 and the spanning tree consisting of the thick arcs, the in-tree arc-fundamental cycle incidence

matrix is given below.

	Fundamental cycle of out-of-tree arc				
	e_5	e_6	e_7	e_8	e_9
In-tree arc e_1	0	0	−1	1	1
e_2	0	−1	1	−1	−1
e_3	−1	1	0	1	1
e_4	1	−1	0	0	−1

For $t = 1$ to m let $E_{.t}$ be the column vector of the node-arc incidence matrix E of G corresponding to the arc e_t. Then from the result in Exercise 1.22 we have

$$E_{.p} = \sum_{t=1}^{n-1} \lambda_{tp} E_{.t} \tag{1.14}$$

for $p = n$ to m. Thus $(\lambda_{1p}, \ldots, \lambda_{n-1,p})$ are the coefficients in the representation of the node-arc incidence vector of out-of-tree arc e_p as a linear combination of the node-arc incidence vectors of in-tree arcs.

For $t = 1$ to $n-1$, let $[\mathbf{X}_t, \bar{\mathbf{X}}_t]$ be the fundamental cutset of the in-tree arc e_t in $\mathbb{T}$. For $t = 1$ to $n-1, p = 1$ to m, define

$$g_{tp} = \begin{cases} 0 & \text{if } e_p \text{ is not in } [\mathbf{X}_t, \bar{\mathbf{X}}_t] \\ +1 & \text{if } e_p \text{ is in } (\mathbf{X}_t, \bar{\mathbf{X}}_t) \\ -1 & \text{if } e_p \text{ is in } (\bar{\mathbf{X}}_t, \mathbf{X}_t) \end{cases}$$

Then $(g_{t1}, \ldots, g_{tm})$ is known as the **fundamental cutset vector corresponding to the in-tree arc** e_t. The $(n-1) \times m$ matrix Q consisting of these rows is known as **fundamental cutset-arc incidence matrix** of G wrt $\mathbb{T}$.

Now consider an undirected network $\mathrm{G} = (\mathcal{N}, \mathcal{A})$ which has no self loops. The **node-edge** or **vertex-edge incidence matrix** of G has a row corresponding to each node and a column corresponding to each edge in G. The entry in the row corresponding to node i and column corresponding to edge e is 1 if e contains i, 0 otherwise. As an example, the node-edge incidence matrix of the network in Figure 1.5 is given below.

Edge →	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Node 1	1	1	1	1	1	0	0
2	1	0	0	1	0	1	0
3	0	1	1	0	0	0	1
4	0	0	0	0	1	1	1

The determinant of the submatrix of this matrix given by rows 1, 2, 4 and columns 4, 5, 6 is -2. So, the node-edge incidence matrix may not be totally unimodular in general.

Exercises

1.27 Let $A = (a_{ij})$, be a matrix with $a_{ij} \in \{-1, 0, 1\}$ for all i, j, in which each column has at most two nonzero entries. Prove that A is totally unimodular iff its rows can be partitioned into two sets so that

(a) If two nonzero elements of a column have the same sign, they are in different sets.

(b) If two nonzero elements of a column have different signs, they are in the same set.
(Heller and Tompkins [1958])

1.28 Show that the determinant of the node-edge incidence matrix of an odd cycle is not in $\{-1, 0, 1\}$. Prove that the node-edge incidence matrix of an undirected network is totally unimodular iff the network is bipartite.

1.29 Let H be an integer matrix of order $m \times n$. Prove that the matrix $A = (H \vdots I)$ where I is the unit matrix of order m, is unimodular iff H is totally unimodular. Thus show that any matrix of order $m \times n$ which contains the unit matrix of order m as a submatrix is unimodular iff it is totally unimodular.

1.30 Let A be an integer matrix of order $m \times n$ and rank r. If $r = m$ and B is a basis for A, prove that A is unimodular iff determinant(B) $= \pm 1$ and the matrix $\bar{A} = B^{-1}A$ is totally unimodular. If $r < m$ let B

be a square nonsingular submatrix of A of order r. Rearrange the rows and columns of A so that this submatrix comes to the top left corner, as in A'.

$$A' = \left(\begin{array}{c|c} B & D \\ \hline F & H \end{array}\right)$$

Prove that A is unimodular iff $(B\vdots D)$ and $(B^T\vdots F^T)$ are unimodular. Since both these matrices are of full row rank, their unimodularity can be characterized in terms of total unimodularity as in the case above. (K. Truemper)

1.31 Let G $= (\mathcal{N}, \mathcal{A})$ be a directed connected network with n nodes and m arcs, and let $\mathbb{T}$ be a spanning tree in G. Let E be the node-arc incidence matrix, and L the fundamental cycle-arc incidence matrix of G wrt $\mathbb{T}$. Do the following: (i) Prove L is totally unimodular and its rank is $m-(n-1)$ (i.e., its rows form a linearly independent set). (ii) Prove $LE^T = 0$. (iii) Prove that the incidence vector (written as a row vector) of any elementary cycle in G can be expressed as a linear combination of the rows of L, with all the coefficients in the expression being either 0,or ± 1. (iv) Prove that a $(m-n+1)\times(m-n+1)$ square submatrix of L is nonsingular iff the columns of this submatrix correspond to a cotree. (v) Let $\mathbb{T}_2$ be a spanning tree in G. Prove that the fundamental cycle-arc incidence matrix of G wrt $\mathbb{T}_2$ is $(L(\bar{\mathbb{T}}_2))^{-1}L$, where $L(\bar{\mathbb{T}}_2)$ is the square submatrix of L consisting of columns in L corresponding to the cotree wrt $\mathbb{T}_2$. (W. Mayeda [1972])

1.32 Let L, Q be the fundamental cycle-arc incidence matrix, and the fundamental cutset-arc incidence matrix respectively, wrt a spanning tree $\mathbb{T}$ in a connected directed network G $= (\mathcal{N}, \mathcal{A})$ with $|\mathcal{N}| = n, |\mathcal{A}| = m$. Do the following: (i) Prove that Q is of full row rank. (ii) Prove that $QL^T = 0, LQ^T = 0$. (iii) Rearrange the arcs in $\mathcal{A}$ so that all the $m-n+1$ out-of-tree arcs wrt $\mathbb{T}$ appear first, and then the $n-1$ in-tree arcs appear. Also, rearrange the columns of Q, L according to this order. Let $Q = (Q_1\vdots Q_2), L = (L_1\vdots L_2)$, after this rearrangement. Columns in Q_1, L_1 correspond to out-of-tree arcs wrt $\mathbb{T}$; and columns in Q_2, L_2 correspond to in-tree arcs. Then prove that $Q_2 = I_{n-1}, L_1 =$

I_{m-n+1}, and $Q_1 = -L_2^T$. (iv) Prove that Q is totally unimodular (use the result in the previous bit) (v) Let $\mathbf{A} \subset \mathcal{A}$, $|\mathbf{A}| = n-1$, and let $Q(\mathbf{A})$ be the submatrix of Q consisting of columns in Q corresponding to arcs in the set $\mathbf{A}$. Prove that $Q(\mathbf{A})$ is nonsingular iff $\mathbf{A}$ is the set of arcs in a spanning tree for G. (vi) Let $\mathbb{T}_2$ be any spanning tree in G and $Q(\mathbb{T}_2)$ be the submatrix of Q consisting of columns in Q corresponding to in-tree arcs in $\mathbb{T}_2$. Prove that the fundamental cutset-arc incidence matrix wrt $\mathbb{T}_2$ is $(Q(\mathbb{T}_2))^{-1}Q$. (vii) Prove L is totally unimodular. Let $\mathbf{A} \subset \mathcal{A}$, $|\mathbf{A}| = m - n + 1$, and let $L(\mathbf{A})$ be the submatrix of L corresponding to arcs in the set $\mathbf{A}$. Prove that $L(\mathbf{A})$ is nonsingular iff $\mathbf{A}$ is a cotree in G.

Matchings and Assignments

Let G $= (\mathcal{N}, \mathcal{A}, c)$ be an undirected network with c as the vector of edge cost coefficients. A **matching** in G is a subset of edges $\mathbf{M} \subset \mathcal{A}$ containing at most one edge incident at any node. For example, in the network in Figure 1.23, the set of thick edges forms a matching. The cost of a matching is defined to be the sum of the cost coefficients of edges in it. A matching is said to be a **perfect matching** if it contains exactly one edge incident at each node. In Figure 1.23, the set of wavy edges is a perfect matching.

Consider an undirected bipartite network with its bipartition $\mathcal{N}_1 = \{R_1, \ldots, R_n\}$, $\mathcal{N}_2 = \{C_1, \ldots, C_n\}$. A perfect matching in this network is called an **assignment** or an **assignment of order n**, it can be represented by a 0-1 square matrix $x = (x_{ij})$ of order n where $x_{ij} = 1$ if the edge $(R_i; C_j)$ is in the assignment, or 0 otherwise. We will also call such matrices as assignments. The x_{ij} in an assignment satisfy the constraints (1.15), (1.16).

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1, \text{ for all } j = 1 \text{ to } n \\ \sum_{j=1}^{n} x_{ij} &= 1, \text{ for all } i = 1 \text{ to } n \\ x_{ij} &\geq 0, \text{ for all } i, j \end{aligned} \tag{1.15}$$

$$\text{and } x_{ij} \quad = \quad 0 \text{ or } 1, \text{ for all } i, j \tag{1.16}$$

Any feasible solution $x = (x_{ij})$ to (1.15) is called a **doubly stochastic matrix**, and an assignment is a 0-1 doubly stochastic matrix (it is also a **permutation matrix**, or a 0-1 square matrix containing a single "1" entry in each row and column). Here is an assignment $\bar{x}$ of order 4.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.17}$$

Let j_r be the column which has the unique "1" entry in row $r, r = 1$ to n, in an assignment x of order n. Then the only nonzero variables in x are x_{r,j_r}, for $r = 1$ to n. In this case we will denote the assignment

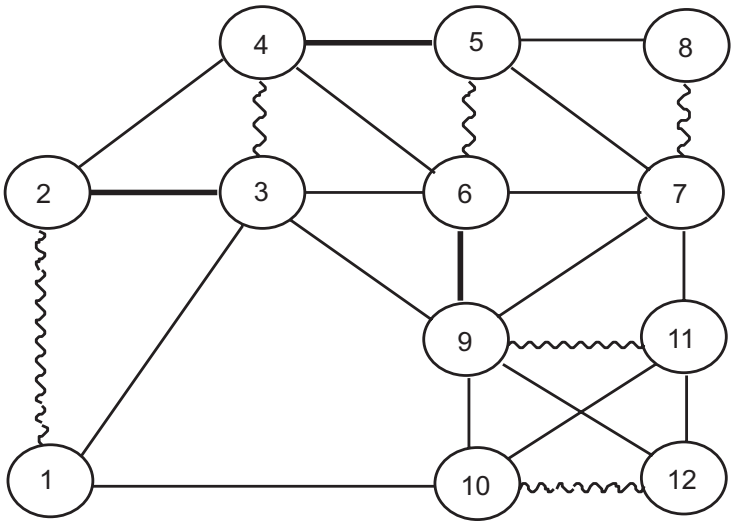


Figure 1.23: The thick edges form a matching in this network. The set of wavy edges is a perfect matching.

x by the set of its unit cells, namely $\{(1, j_1), \ldots, (n, j_n)\}$. In this notation, the assignment $\bar{x}$ in (1.17) will be denoted by $\{(1, 2), (2, 4), (3, 1), (4, 3)\}$.

Assignments are useful to model situations in which there are two distinct sets of objects of equal numbers, and we need to form them into pairs, each pair consisting of one object of each set. For example, $\mathcal{N}_1$ might be a set of n boys, and $\mathcal{N}_2$ a set of an equal number of girls. If each boy in $\mathcal{N}_1$ marries a girl in $\mathcal{N}_2$, the resulting set of couples will be an assignment. Sometimes we will take both the sets $\mathcal{N}_1$, $\mathcal{N}_2$, to be $\{1, \ldots, n\}$. It should be understood that they refer to the serial numbers of distinct sets of objects.

1.2.3 Single Commodity Node-Arc Flow Models

In these models, it is assumed that the flow of the material is carried out independently along each line of the network. Consider the directed flow network G $= (\mathcal{N}, \mathcal{A}, \ell, k, \breve{s}, \breve{t})$. The variable f_{ij} represents the amount of material transported from i to j along the arc $(i, j) \in \mathcal{A}$, and the vector $f = (f_{ij})$ is known as the **node-arc flow vector**, or just the **flow vector**.

The quantity $\sum_{j \in \mathbf{B}(i)} f_{ji} = f(\mathcal{N}, i)$ is the total amount of material flowing into node i, and $\sum_{j \in \mathbf{A}(i)} f_{ij} = f(i, \mathcal{N})$ is that flowing out of node i, these two quantities must be equal if i is an intermediate node. The net amount of material leaving the source node $\breve{s}$, $f(\breve{s}, \mathcal{N}) - f(\mathcal{N}, \breve{s})$, is known as the **value of the flow vector** f, and denoted by $v(f)$ or v. Since all the material leaving the source has to eventually reach the sink, the net amount reaching the sink, $f(\mathcal{N}, \breve{t}) - f(\breve{t}, \mathcal{N})$ should be equal to v too. So, a flow vector f is a **feasible flow vector** in G if it satisfies the following constraints:

$$f(i, \mathcal{N}) - f(\mathcal{N}, i) = \begin{cases} v & \text{if } i \text{ is source} \\ -v & \text{if } i \text{ is sink} \\ 0 & \text{if } i \text{ is an intermediate node} \end{cases} \tag{1.18}$$

$$\text{and } \ell \leq f \leq k \tag{1.19}$$

Unless otherwise specified, we will assume that the source node has an unlimited amount to be shipped out and that the sink can receive an unlimited amount. The constraints in (1.18) are known as **flow conservation equations**. For any arc (i, j), the associated variable

f_{ij} appears in exactly two equations in (1.18); once with a coefficient of + 1 in the equation corresponding to node i, and another time with a coefficient of -1 in the equation corresponding to node j. Hence, the coefficient matrix of the flow variables f_{ij} in (1.18) is the node-arc incidence matrix of G. As an example, consider the network in Figure 1.24. The flow conservation equations for this network are shown in the table given below. Verify that the flow vector marked in Figure 1.24 is feasible (e.g.,total amount reaching node 5 is 6 units along (3, 5). Total amount leaving node 5 is 6 units, 2 along (5, 2) and 4 along (5, 4). So conservation holds at node 5, etc.) This flow vector has a value of 7 units.

	f_{12}	f_{13}	f_{23}	f_{24}	f_{52}	f_{34}	f_{35}	f_{54}	f_{46}	f_{56}	$-v$	
Node 1	1	1									1	0
2	-1		1	1	-1							0
3		-1	-1			1	1					0
4				-1		-1		-1	1			0
5					1		-1	1		1		0
6									-1	-1	-1	0

Verify that the coefficient matrix of the flow variables in the conservation equations is the node-arc incidence matrix E of G. If q denotes the node-arc incidence vector of the hypothetical $(\breve{s}, \breve{t})$ arc, the constraints on f for feasibility are: $Ef - qv = 0$, and $\ell \leq f \leq k$. The **maximum value flow problem** in G is to maximize v subject to these constraints.

If $f = (f_{ij})$ is a feasible flow vector in G, for each $(i, j) \in \mathcal{A}, k_{ij} - f_{ij}$ is known as the **residual capacity of arc** (i, j) wrt f. The arc (i, j) is said to be **saturated** in f if its residual capacity is 0, i.e., if $f_{ij} = k_{ij}$.

In general, there may be several source nodes where material is available, and several sink nodes where it is required. The **exogenous flow** at a node, or flow that is external to the network, refers to the quantity of such material at these nodes. At node i the exogenous flow will be denoted by V_i, the convention is that if $V_i > 0$, then i is a source with V_i units available; and if $V_i < 0, i$ is a sink with a requirement of $|V_i|$ units. If $V_i = 0$, there is no exogenous flow at i, and it is a

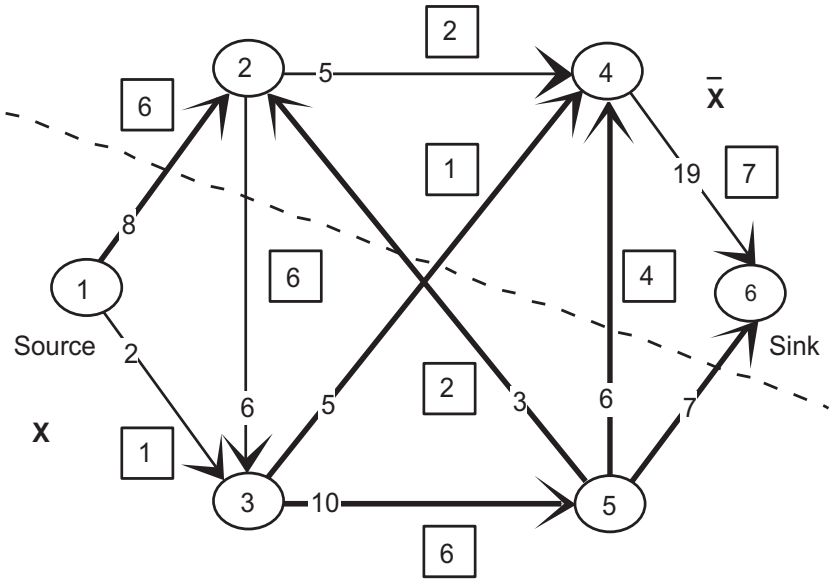


Figure 1.24: Capacities are entered in the gaps on the arcs. Lower bounds are all 0. If flow on an arc is nonzero, it is entered inside a little square by its side.

transit node. Sometimes $-V_i$ is called the **requirement** at node i. The vector $V = (V_i)$ leads to the right-hand side constants vector in LP formulations of network flow problems.

When there are several source and sink nodes in a single commodity flow problem, we assume that the material from any source can be shipped to any sink. Otherwise, if some sources can only ship to certain sinks, it becomes necessary to keep track of the flows from each source separately, and the problem becomes a **multicommodity flow problem**.

The general LP type problem in the connected directed single commodity flow network $G = (\mathcal{N}, \mathcal{A}, \ell, k, V)$, with node-arc incidence matrix E is

$$\begin{aligned} &\text{Minimize } cf \\ &\text{Subject to } Ef \;=\; V \end{aligned} \tag{1.20}$$

$$\ell \leq f \leq k$$

Since the sum of all the row vectors of E is 0, a necessary condition for the existence of a feasible flow vector in G is:

$$\sum_{i \in \mathcal{N}} V_i = 0 \tag{1.21}$$

When G is connected, from the results in the previous section it is clear that the system (1.20) contains exactly one redundant equality constraint. Any one of the equality constraints in (1.20) may be treated as a redundant constraint and eliminated, making the remaining system nonredundant.

Let $|\mathcal{N}| = n, |\mathcal{A}| = m$. Number the arcs in G as $e_t, t = 1$ to m, and denote the flow amount on e_t by f_t. Select any node, say node n, and delete the equality constraint corresponding to it from (1.20). Let the remaining system of equations there be

$$\bar{E}f = \bar{V} \tag{1.22}$$

Since G is connected, $\bar{E}$ is of full row rank. From Theorem 1.4 we know that every basis for $\bar{E}$ is triangular. The result in Corollary 1.1 implies that every basic solution of (1.22) is an integer vector if ℓ, k, V are integer vectors. If an LP has an optimum solution it has one which is a BFS. So, we conclude that if the data in an LP type single commodity pure network flow problem is integer, and it has an optimum solution, then it has one in which all the flow amounts are integer. This includes the maximum value flow problem, and the minimum cost flow problem (1.20).

From the results in Section 1.2.2, we know that the set of columns of $\bar{E}$ associated with the arcs in a cycle is linearly dependent and that associated with a forest in G is linearly independent. These facts imply that every basic vector for (1.22) consists of node-arc flow variables associated with arcs in a spanning tree in G and vice versa. Let B denote a basis for (1.22) with its basic columns corresponding to the in-tree arcs, $e_1, \ldots, e_{n-1}$ of a spanning tree $\mathbb{T}$ in G. Let D be the matrix consisting of the nonbasic columns. When partitioned into basic, nonbasic parts this way, (1.22) becomes

$f_1, \ldots, f_{n-1}$	$f_n, \ldots, f_m$	
B	D	$\bar{V}$

Select node n as the root node for $\mathbb{T}$. Let $(\lambda_{1p}, \ldots, \lambda_{mp})$ be the incidence vector of the fundamental cycle of the out-of-tree arc $e_p, p = n$ to m, and let λ be the in-tree arc-fundamental cycle incidence matrix. Let $\bar{E}_{.t}$ denote the column of $\bar{E}$ associated with the arc $e_t, t = 1$ to m. Then from (1.14) we have $\bar{E}_{.p} = \sum_{t=1}^{n-1} \lambda_{tp} \bar{E}_{.t}$, for $p = n$ to m. This implies that $D = B\lambda$, or $B^{-1}D = \lambda$. Hence the canonical tableau of (1.22) wrt the basis B is:

$f_1, \ldots, f_{n-1}$	$f_n, \ldots, f_m$	
I	λ	$B^{-1}\bar{V}$

So, the updated nonbasic part in the canonical tableau of (1.22) is always the in-tree arc-fundamental cycle incidence matrix wrt the spanning tree consisting of the basic arcs, and hence can be constructed combinatorially without the need to perform any pivot steps. This also leads to the following property.

DANTZIG PROPERTY In every canonical tableau for the system of conservation equations in a single commodity flow problem in a pure network, all the entries are always 0, or ± 1.

As an example consider the network in Figure 1.20. Select node 5 as the root node and eliminate the equation corresponding to it from the system of conservation equations for this network. This leads to the next tableau.

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	
1	0	0	0	0	0	−1	1	1	V_1
−1	−1	0	0	0	1	0	0	0	V_2
0	1	1	0	−1	0	1	0	0	V_3
0	0	0	1	1	−1	0	0	−1	V_4

For this system (f_1, f_2, f_3, f_4) is a basic vector. It corresponds to the spanning tree with thick arcs in Figure 1.20. The canonical tableau of this system wrt this basic vector is:

Basic variable	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	
f_1	1	0	0	0	0	0	−1	1	1	$\hat{V}_1$
f_2	0	1	0	0	0	−1	1	−1	−1	$\hat{V}_2$
f_3	0	0	1	0	−1	1	0	1	1	$\hat{V}_3$
f_4	0	0	0	1	1	−1	0	0	−1	$\hat{V}_4$

It can be verified that the updated nonbasic part under the nonbasic variables f_5 to f_9 is exactly the in-tree arc-fundamental cycle incidence matrix derived earlier in Section 1.2.2.

Computing the Inverse of a Basis for $\bar{E}$ Combinatorially

Let B be a basis for (1.22). Suppose it corresponds to the spanning tree $\mathbb{T}$ with in-tree arcs $e_1, \ldots, e_{n-1}$. Here we discuss how to compute B^{-1} combinatorially without the need to perform any pivot steps. Introduce artificial arcs $e_{m+t} = (t, n), t = 1$ to $n-1$, joining each nonroot node to the root node n. With these artificial arcs, the node-arc incidence matrix of the augmented network has the following form:

Row corresponding to node	In-tree arcs $e_1, \ldots, e_{n-1}$	Out-of-tree arcs	Artificial arcs $e_{m+1}, \ldots, e_{m+n+1}$
1 2 ⋮	B	D	I_{n-1}
Root node n	...	...	$-1 \ldots -1$

From this table, and the above discussion, it is clear that the in-tree arc-fundamental cycle incidence matrix of the artificial arcs is B^{-1}, and B^{-1} can therefore be computed directly from the network.

As an example, consider the basis B given by the submatrix of the node-arc incidence matrix of the network in Figure 1.20, corresponding to nodes 1 to 4 and arcs $e_1, \ldots, e_4$. To find B^{-1}, include artificial arcs $(i, 5), i = 1$ to 4. Figure 1.25 just depicts the in-tree arcs and the artificial arcs which are dashed.

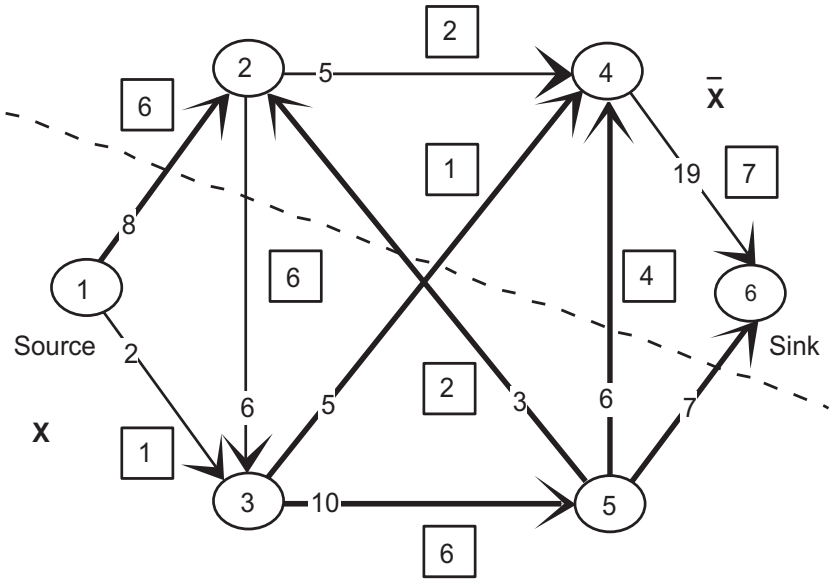


Figure 1.25:

The remaining matrix, after eliminating the row corresponding to root node 5 from the node-arc incidence matrix of the network in Figure 1.25, is $(B\vdots I)$ given below. The in-tree arc-fundamental cycle incidence matrix for the network in Figure 1.25 is given next and it can be verified that it is B^{-1}. This is followed by Theorem 1.8 stating a fundamental property of basis inverses in pure network flow problems. This result is used in Chapter 5 to develop purely combinatorial techniques for resolving the problem of cycling under degeneracy in the primal simplex algorithm for single commodity pure network flow problems, without the need for maintaining B^{-1}.

Basic arcs				Artificial arcs			
e_1	e_2	e_3	e_4	(1, 5)	(2, 5)	(3, 5)	(4, 5)
1	0	0	0	1	0	0	0
−1	−1	0	0	0	1	0	0
0	1	1	0	0	0	1	0
0	0	0	1	0	0	0	1

	Fundamental cycle of artificial arc			
In-tree arc e_1	1	0	0	0
e_2	−1	−1	0	0
e_3	1	1	1	0
e_4	0	0	0	1

Unisign Property of Rows of Basis Inverses in Pure Network Flow Problems

THEOREM 1.8 *Let $\bar{E}$ be the matrix remaining after the row corresponding to the root node n is deleted from the node-arc incidence matrix E of a directed connected network $\mathrm{G} = (\mathcal{N}, \mathcal{A})$. Let B be a basis for $\bar{E}$ associated with the spanning tree $\mathbb{T}$ with its columns corresponding to in-tree arcs $e_1, \ldots, e_{n-1}$, say, in that order. Then all the nonzero entries in $(B^{-1})_{t\cdot}$ are -1 if e_t is directed away from the root node, or $+1$ if e_t is directed towards the root node, in $\mathbb{T}$.*

Proof From the procedure for computing B^{-1} discussed above, we see that the jth entry in $(B^{-1})_{t\cdot}$ is: 0 if e_t is not on the fundamental cycle of (j, n); $+1$ $[-1]$ if e_t is on this fundamental cycle with an orientation opposite to [same as] that of (j, n) (i.e., if e_t is directed towards [away from] the root node n). This clearly implies the result in the theorem. ∎

Optimality Conditions

Consider the minimum cost flow problem (1.20) in the directed single commodity flow network $\mathrm{G} = (\mathcal{N}, \mathcal{A}, \ell, k, c, V)$. In the dual problem there are dual variables π_i associated with the flow conservation equation at node i, they are called **dual variables**, or **node prices**, or **node potentials**. The complementary slackness optimality conditions for a feasible flow vector f are the existence of a dual vector $\pi = (\pi_i)$ satisfying: for each $(i, j) \in \mathcal{A}$, if

$$\begin{array}{lll} \pi_j - \pi_i > c_{ij} & \text{then} & f_{ij} = k_{ij} \\ \pi_j - \pi_i < c_{ij} & \text{then} & f_{ij} = \ell_{ij} \\ \pi_j - \pi_i = c_{ij} & \text{then} & \ell_{ij} \leq f_{ij} \leq k_{ij} \end{array}$$

The potential difference $\pi_j - \pi_i$ is known as the **tension** across the arc (i, j) in the node potential vector π, the optimality conditions depend only on these tensions, and not directly on the potentials themselves. Two different node potential vectors that differ by a constant give rise to the same tension vector. The tension vector is actually $-\pi E$ where E is the node-arc incidence matrix. The quantity $\bar{c}_{ij} = c_{ij} - (\pi_j - \pi_i)$ is known as the **reduced cost coefficient** of arc (i, j) wrt π.

Flow Augmenting Paths

Let $f = (f_{ij})$ be a feasible flow vector of value v in the directed single commodity flow network $\mathrm{G} = (\mathcal{N}, \mathcal{A}, \ell, k, \breve{s}, \breve{t})$. A path $\mathcal{P}$ from $\breve{s}$ to $\breve{t}$ is said to be a **flow augmenting path (FAP)** wrt f if it satisfies

$$f_{ij} \begin{cases} < k_{ij} & \text{for forward arcs } (i,j) \text{ on } \mathcal{P} \\ > \ell_{ij} & \text{for reverse arcs } (i,j) \text{ on } \mathcal{P} \end{cases}$$

The reason for this name can be easily explained. Let $\epsilon = \min \{\epsilon_1, \epsilon_2\}$ where $\epsilon_1 = \min \{ (k_{ij} - f_{ij}) : (i,j) \text{ a forward arc on } \mathcal{P} \}$, $\epsilon_2 = \min \{ (f_{ij} - \ell_{ij}) : (i,j) \text{ a reverse arc on } \mathcal{P} \}$. $\epsilon_1(\epsilon_2)$ is defined to be $+\infty$ if there are no forward (reverse) arcs on $\mathcal{P}$. ϵ is called the **residual capacity** of the FAP $\mathcal{P}$, it is > 0. Define a new flow vector $\hat{f} = (\hat{f}_{ij})$ by

$$\hat{f}_{ij} = \begin{cases} f_{ij} & \text{if } (i,j) \text{ is not on } \mathcal{P} \\ f_{ij} + \epsilon & \text{if } (i,j) \text{ is a forward arc on } \mathcal{P} \\ f_{ij} - \epsilon & \text{if } (i,j) \text{ is a reverse arc on } \mathcal{P} \end{cases}$$

Then, $\hat{f}$ is a feasible flow vector of value $\hat{v} = v + \epsilon$. This operation of computing $\hat{f}$ from f is called the **flow augmentation step** using

the FAP $\mathcal{P}$. After this step, $\mathcal{P}$ is no longer an FAP wrt the new flow vector $\hat{f}$.

As an example, consider the feasible flow vector f in Figure 1.24. The thick path from source to sink with forward arcs (1, 2), (5, 4), (3, 5), (5, 6), and reverse arcs (5, 2), (3, 4) is an FAP wrt f. $\epsilon_1 = \min \{8-6, 6-4, 10-6, 7-0\} = 2$, $\epsilon_2 = \min \{2-0, 1-0\} = 1$. $\epsilon = \min \{2, 1\} = 1$. The new flow vector obtained after flow augmentation is $\hat{f} = (\hat{f}_{12}, \hat{f}_{13}, \hat{f}_{23}, \hat{f}_{24}, \hat{f}_{52}, \hat{f}_{34}, \hat{f}_{35}, \hat{f}_{54}, \hat{f}_{46}, \hat{f}_{56}) = (7, 1, 6, 2, 1, 0, 7, 5, 7, 1)$. It has value 8. This example indicates that an FAP need not be a simple path. However, in the labeling algorithms discussed in Chapter 2, all FAPs identified will be simple paths.

Several of the algorithms discussed in later chapters use subroutines which generate FAPs, and hence they are called **augmenting path methods**.

An FAP is said to be a **flow augmenting chain, FAC**, if it is a chain (i.e., if all the arcs on it are forward arcs).

A feasible flow vector of value v in G is said to be a **maximum value feasible flow vector** if v is the maximum value attainable in G, or a **maximal** or **blocking feasible flow vector** if there exists no FAC wrt it.

EXAMPLE 1.3

Every maximum value flow vector is maximal, but the converse may not be true. Figure 1.26 illustrates this point.

Residual Cycles and the Residual Networks G(f), G(f, π)

Let $f = (f_{ij})$ be a flow vector in a directed, connected, single commodity flow network G $= (\mathcal{N}, \mathcal{A}, \ell, k, c, \check{s}, \check{t})$ satisfying the bound conditions on all the flow variables (i.e., $\ell \leq f \leq k$), but may or may not satisfy flow conservation at the nodes.

If an arc $(i, j) \in \mathcal{A}$ satisfies $f_{ij} < k_{ij}$, the flow amount on it can be increased by an amount equal to its residual capacity of $\kappa_{ij} = k_{ij} - f_{ij}$ without violating the upper bound, hence it is called a **residual arc**

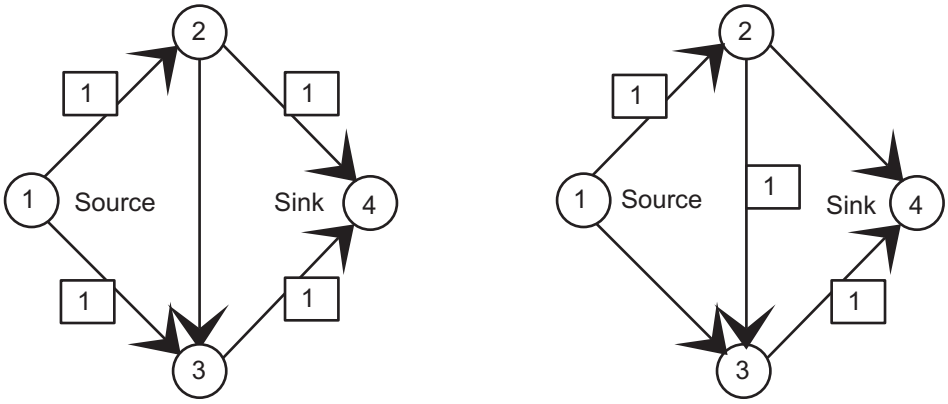


Figure 1.26: Two copies of a network. Each arc has lower bound 0 and capacity 1. The flow on an arc is entered in a box by its side if it is nonzero. The one on the left is a maximum value feasible flow vector of value 2. The one on the right has value 1. It is a maximal flow, but not of maximum value.

wrt f. The cost coefficient of this residual arc is naturally c_{ij}, since it is the unit cost of increasing the flow on (i, j).

Likewise, if $(p, q) \in \mathcal{A}$ satisfies $f_{pq} > \ell_{pq}$, the flow on it can be decreased by an amount equal to $f_{pq} - \ell_{pq}$ without violating the lower bound. This is equivalent to creating a new arc (q, p), which may not be an arc in G, and increasing the flow on it from 0 by the same amount. So, in this case, we call (q, p) a residual arc wrt f, and associate the cost coefficient $-c_{pq}$ with it, since increasing the flow on it is the same as decreasing the flow on the original arc (p, q). Construct the set of arcs $\mathcal{A}(f)$ by the following rules.

1. For each $(i, j) \in \mathcal{A}$ satisfying $f_{ij} < k_{ij}$ include the arc (i, j) in $\mathcal{A}(f)$ with a + label, lower bound 0, capacity $\kappa_{ij} = k_{ij} - f_{ij}$, and cost coefficient $c'_{ij} = c_{ij}$.

2. For each $(i, j) \in \mathcal{A}$ satisfying $f_{ij} > \ell_{ij}$ include the arc (j, i) in $\mathcal{A}(f)$ with a − label, lower bound 0, capacity $\kappa_{ji} = f_{ij} - \ell_{ij}$, and cost coefficient $c'_{ji} = -c_{ij}$.

$\mathcal{A}(f)$ is the set of **residual arcs** and $G(f) = (\mathcal{N}, \mathcal{A}(f), 0, \kappa, c')$ is the **residual network** wrt f, it is very useful for determining flow vectors that we can move to from f while maintaining bound feasibility. Each arc in $G(f)$ corresponds to an arc in G. $(p, q) \in \mathcal{A}(f)$ corresponds to (p, q) in G if its label is $+$, or to (q, p) in G if its label is $-$. Under this correspondence, every FAP from $\breve{s}$ to $\breve{t}$ wrt f corresponds to a chain from $\breve{s}$ to $\breve{t}$ in $G(f)$ and vice versa. Let $\mathcal{C}$ be any chain in $G(f)$ and $\mathcal{P}$ the path corresponding to it in G. If $\epsilon = \min \{\kappa_{ij} : (i, j)$ is an arc on $\mathcal{C}\}$, then $\epsilon > 0$. In f, increase (decrease) the flow on all forward (reverse) arcs of $\mathcal{P}$ by ϵ, this leads to a new flow vector in G which also satisfies the bounds.

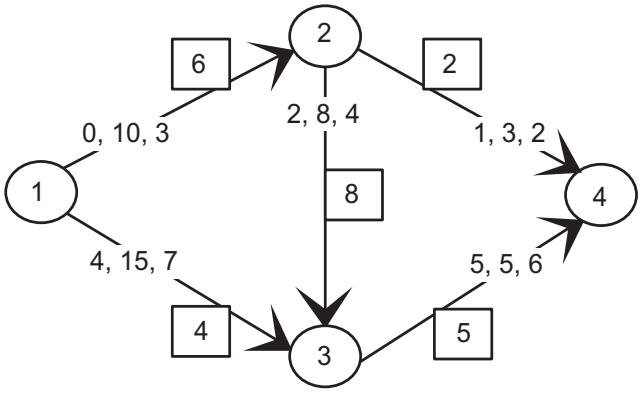


Figure 1.27: The network G and a bound feasible flow vector f in it.

When the residual network is used in the study of the maximum value flow problem, no cost coefficients are used since there are no costs in this problem.

As an example, consider the network G in Figure 1.27. On each arc, the lower bound, capacity, and cost coefficient are entered in that order, and a bound feasible (but conservation violating) flow vector f is marked in little squares. The residual network $G(f)$ is given in Figure 1.28 with the arc labels $+$ or $-$ entered.

Suppose we are given a bound feasible flow vector f, and a node price vector $\pi = (\pi_i)$ in G. $\bar{c}_{ij} = c_{ij} - (\pi_j - \pi_i)$ is the reduced cost coefficient of arc $(i, j) \in \mathcal{A}$. The **residual network** wrt f, π, $G(f, \pi)$ $= (\mathcal{N}, \mathcal{A}(f), 0, \kappa, \bar{c}')$ is the same as $G(f)$, with the exception that the

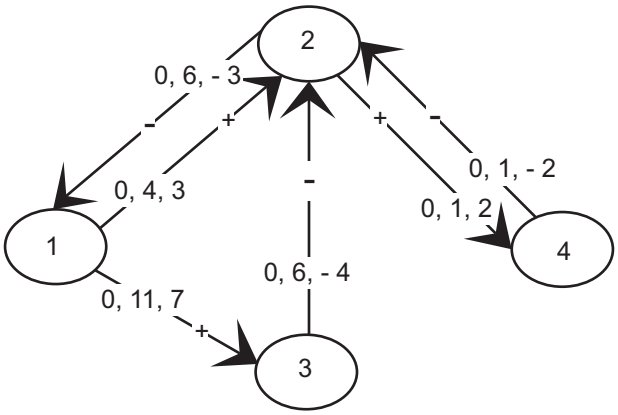


Figure 1.28: The residual network G(f).

arc cost coefficients in it are determined using $\bar{c}$ as the cost vector in G instead of c.

An oriented cycle $\mathbb{C}$ in G is said to be a **residual cycle** wrt the flow vector $f = (f_{ij})$ satisfying the bound conditions on all the flow variables, if $f_{ij} < k_{ij}$ on all forward arcs on $\mathbb{C}$, and $f_{ij} > \ell_{ij}$ on all reverse arcs in $\mathbb{C}$. The **capacity of this residual cycle** is defined to be min. $\{ k_{ij} - f_{ij} : (i,j)$ a forward arc on $\mathbb{C} \} \cup \{f_{ij} - \ell_{ij} : (i,j)$ a reverse arc on $\mathbb{C}\}$. For example, in the network in Figure 1.27, the oriented cycle 1, (1, 3), 3, (2, 3), 2, (1, 2),1 oriented in the anticlockwise direction, is a residual cycle wrt the flow vector marked there, of capacity 6. Notice that every residual cycle wrt f corresponds to a simple circuit in the residual network and vice versa.

Feasible Circulations

A flow vector $f = (f_{ij})$ in a directed single commodity flow network G $= (\mathcal{N}, \mathcal{A}, \ell, k)$ is said to be a **feasible circulation** if it satisfies the bounds, $\ell \leq f \leq k$, and $f(i, \mathcal{N}) - f(\mathcal{N}, i) = 0$ for all $i \in \mathcal{N}$ (this is the same as $Ef = 0$, where E is the node-arc incidence matrix of G). See Figure 1.29 for an illustration.

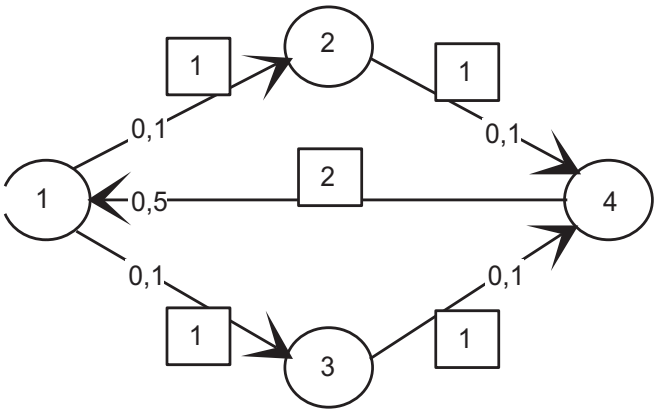


Figure 1.29: Data on the arcs is lower bound, capacity in that order; and flow amount in a box if it is nonzero.

1.2.4 The Arc-Chain Flow Model

Let G $= (\mathcal{N}, \mathcal{A}, \ell = 0, k, \breve{s}, \breve{t})$ be a directed single commodity flow network. In this model we determine various chains from $\breve{s}$ to $\breve{t}$, and specify how much to ship directly from $\breve{s}$ to $\breve{t}$ across each of them. The chains considered may have common arcs. Let $\mathcal{C}_1, \ldots, \mathcal{C}_P$ be all the distinct chains from $\breve{s}$ to $\breve{t}$ in G, and let x_h denote the amount of material shipped along $\mathcal{C}_h$, $h = 1$ to P. The x_h are the decision variables in this model, and $x = (x_h)$ is the arc-chain flow vector. Here, each x_h has to be clearly associated with its chain $\mathcal{C}_h$. P is likely to be large, and in specifying this vector x, it is only necessary to list the positive x_h in it, and the chains associated with them.

The **value of the arc-chain flow vector** $x = (x_h)$ (i.e., the amount of material reaching the sink in it), is clearly $\sum_h x_h$. Let $\mathbf{C}(i, j)$ denote the set of all the chains from $\breve{s}$ to $\breve{t}$ which contain the arc (i, j). We will use the same symbol to represent the set of their indices. If the arc-chain flow vector $x = (x_h)$ is implemented, the total amount of this flow passing through arc (i, j) will be $\sum(x_h : \text{over } h \in \mathbf{C}(i, j))$. This has to be $\leq k_{ij}$ for feasibility. So, the arc-chain formulation of the maximum value flow problem in G is

$$\begin{aligned} \text{Maximize} \sum_h x_h & \\ \text{Subject to} \sum_{h \in \mathbf{C}(i,j)} x_h & \leq k_{ij} \text{ for each } (i,j) \in \mathcal{A} \\ x_h & \geq 0 \text{ for all } h \end{aligned} \tag{1.23}$$

Given an arc-chain flow vector $x = (x_h)$, in G of value v, define for each $(i,j) \in \mathcal{A}, f_{ij}(x) = \sum(x_h : \text{ over } h \in \mathbf{C}(i,j))$, and $f(x) = (f_{ij}(x) : (i,j) \in \mathcal{A})$. Then it can be verified that $f(x)$ is a feasible node-arc flow vector of value v It is the natural and unique node-arc flow vector corresponding to the arc-chain flow vector x. Both have the same value.

The reverse question is: Given a node-arc feasible flow vector f in G, is there an arc-chain flow vector corresponding to it? We now study this question and the relationship between the two flow models.

THEOREM 1.9 *Let $\tilde{f} = (\tilde{f}_{ij})$ be a feasible node-arc flow vector in G of value $\tilde{v}$. If $\tilde{v} > 0$, there exists a chain from $\breve{s}$ to $\breve{t}$ in G such that $\tilde{f}_{ij} > 0$ on all arcs (i,j) along this chain.*

Proof Assume that $\tilde{v} > 0$. Call an arc $(i,j) \in \mathcal{A}$ a P-arc if $\tilde{f}_{ij} > 0$. We need to show that there is a chain from $\breve{s}$ to $\breve{t}$ consisting of P-arcs only. Define $\mathbf{X} = \{ x : x \in \mathcal{N}$, there exists a chain from $\breve{s}$ to x with P-arcs only$\}$. So, we need to show that $\breve{t} \in \mathbf{X}$.

If $\mathcal{C}_1$ is a chain from $\breve{s}$ to x with P-arcs only, and (x,y) is a P-arc, by including (x,y) at the end of $\mathcal{C}_1$ we get a chain from $\breve{s}$ to y with P-arcs only. This observation leads to the following tree growth scheme to determine the set $\mathbf{X}$. $\breve{s}$ is the root node. The list is always the present set of labeled and unscanned nodes. Each labeled node is in the set $\mathbf{X}$. Since we are only interested in showing that $\breve{t}$ is in the set $\mathbf{X}$, we will terminate the scheme whenever $\breve{t}$ is labeled.

Step 1 Label $\breve{s}$ with $\emptyset$. All other nodes are unlabeled. List $= \{ \breve{s} \}$.

Step 2 If list $= \emptyset$, terminate. Otherwise select a node from it, say i to scan as below. Delete i from list. Find $\mathbf{J} = \{ j$: j unlabeled so far

and (i, j) is a P-arc}. Label each node in **J** with its predecessor index i, and include all of them in the list.

Step 3 If $\breve{t}$ is labeled, find its predecessor path by a backwards trace. This path, written in reverse order beginning with $\breve{s}$ is a chain to $\breve{t}$ with P-arcs only. Terminate.

If $\breve{t}$ is unlabeled, go back to Step 2.

Suppose this scheme terminates without $\breve{t}$ ever getting labeled. Then, $\mathbf{X}$ = set of labeled nodes at this stage. Let $\bar{\mathbf{X}} = \mathcal{N} \backslash \mathbf{X}$. Since the scheme has terminated, from the labeling rules used, we have $\tilde{f}_{ij} = 0$ for all $(i, j) \in \mathcal{A}$ with $i \in \mathbf{X}$, $j \in \bar{\mathbf{X}}$, so $\tilde{f}(\mathbf{X}, \bar{\mathbf{X}}) = 0$. From the conservation equations, we have

$$\tilde{f}(i, \mathcal{N}) - \tilde{f}(\mathcal{N}, i) = \begin{cases} \tilde{v} & \text{for } i = \breve{s} \\ 0 & \text{for } i \neq \breve{s} \text{ or } \breve{t} \end{cases}$$

Summing these over $i \in \mathbf{X}$, we get $\tilde{v} = \tilde{f}(\mathbf{X}, \mathcal{N}) - \tilde{f}(\mathcal{N}, \mathbf{X}) = \tilde{f}(\mathbf{X}, \bar{\mathbf{X}})$ - $\tilde{f}(\bar{\mathbf{X}}, \mathbf{X}) = -\tilde{f}(\bar{\mathbf{X}}, \mathbf{X})$, since $\tilde{f}(\mathbf{X}, \bar{\mathbf{X}}) = 0$. Since all $\tilde{f}_{ij} \geq 0$, this implies that $\tilde{v} = -\tilde{f}(\bar{\mathbf{X}}, \mathbf{X}) \leq 0$, a contradiction to the hypothesis that $\tilde{v} > 0$. Hence it is impossible for the above scheme to terminate without $\breve{t}$ getting labeled. So, it would terminate in Step 3 by producing a chain from $\breve{s}$ to $\breve{t}$ with all arcs on it satisfying $\tilde{f}_{ij} > 0$. ∎

As an example consider the network in Figure 1.30, and the feasible node-arc flow vector of value 24 marked in it. The above scheme is applied on it leading to the predecessor indices entered by the side of the nodes. The sink, node 4 is labeled. Its predecessor path written in reverse order, yields the chain $\mathcal{C}_1$ = 1, (1, 2), 2, (2, 4), 4 in which all the arcs are carrying positive flow.

If the above scheme is operated without Step 3, the set of labeled nodes at termination will be the set **X** defined above.

We can use the following procedure based on the scheme in the proof of Theorem 1.9 to obtain an arc-chain flow corresponding to a given node-arc flow $\tilde{f}$ of value $\tilde{v}$ in G. If $\tilde{v} < 0$ (this can happen if there are some arcs incident into $\breve{s}$ in G, and they carry positive flow) it would imply that in the flow vector $\tilde{f}$, actually $|\tilde{v}|$ units of material

is flowing back from $\breve{t}$ to $\breve{s}$. So, if $\tilde{v} \leq 0$, define the arc-chain flow vector corresponding to $\tilde{f}$ to be $\tilde{x} = 0$, and terminate. On the other hand, if $\tilde{v} > 0$, find a chain from $\breve{s}$ to $\breve{t}$, $\mathcal{C}_1$ say, with positive flows in $\tilde{f}$ on all its arcs. Define the arc-chain flow amount on $\mathcal{C}_1$ to be $\tilde{x}_1$ = min $\{ \tilde{f}_{ij} : (i,j)$ an arc on $\mathcal{C}_1 \}$. In $\tilde{f}$ subtract $\tilde{x}_1$ from the flow amounts on all the arcs of $\mathcal{C}_1$ leading to the new node-arc flow vector $\hat{f}$ say. $\hat{f}$ is a feasible node-arc flow vector in G of value $\hat{v} = \tilde{v} - \tilde{x}_1$. If $\hat{v} \leq 0$ terminate, otherwise repeat this step with $\hat{f}$ and continue in the same way. The arc-chain flow amounts on the chains obtained until termination define an arc-chain vector whose value is $\geq \tilde{v}$.

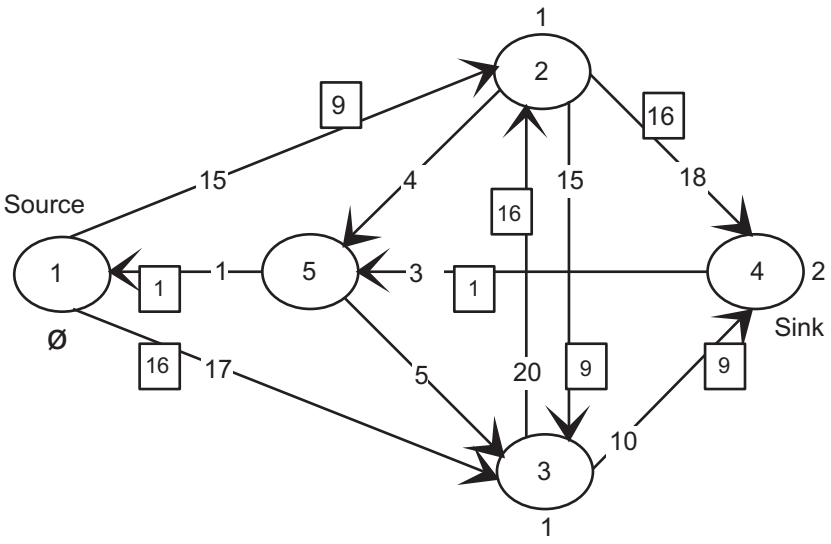


Figure 1.30: Capacities are marked on the arcs and node-arc flow amounts entered inside a box by the side of the arc. Predecessor labels are entered by the side of the nodes. Scheme terminated when the sink is labeled with 2.

For an example, we consider the node-arc flow given in Figure 1.30 of value 24. We take the first chain to be $\mathcal{C}_1$ [consisting of arcs (1, 2) and (2, 4)] obtained above, and the arc-chain flow amount on it is min $\{9, 16\} = 9 = \tilde{x}_1$. We modify the node-arc flow vector by subtracting 9 from the flows on arcs (1, 2), (2, 4) and continue the same way. We obtain the chains $\mathcal{C}_2$ [consisting of arcs (1, 3), (3, 4)] with an arc-chain

flow amount $\tilde{x}_2 = 9$, $\mathcal{C}_3$ [consisting of arcs (1, 3), (3, 2), (2, 4)] with an arc-chain flow amount $\tilde{x}_3 = 7$, and the procedure then terminates. So, the arc-chain flow vector obtained has positive flows on three chains only, and its value is 9+9+7 = 25.

The arc-chain flow vector obtained by this procedure depends on the chains obtained and the order in which they are obtained in the procedure.

THEOREM 1.10 *The maximum flow value from $\breve{s}$ to $\breve{t}$ in the directed single commodity flow network* $\mathrm{G} = (\mathcal{N}, \mathcal{A}, \ell = 0, k, \breve{s}, \breve{t})$ *is the same, irrespective of whether it is modeled using the node-arc or arc-chain flow models.*

Proof Since $\ell = 0$, $f = 0$ is a feasible flow vector of value 0. Therefore, the maximum flow value in G is ≥ 0. Given a maximum value feasible flow in G in either model, the methods discussed above can be used to construct a corresponding flow of at least the same value in the other model. This proves the theorem. ∎

The arc-chain formulation typically has too many variables. It is practical only under a method which operates by maintaining a small set of chains on which the arc-chain flow amount is positive, and generates new chains to introduce into it one by one as necessary. Since each chain corresponds to a column in the model, such approaches are called **column generation approaches**; they are used together with the revised simplex method. However, for single commodity flow problems, the node-arc formulation leads to algorithms which are much more efficient, and this model is therefore used commonly. For multicommodity flow problems the arc-chain formulation leads to a reasonable solution approach. This is discussed in Section 5.11.

Exercises

1.33 Consider the single commodity flow network in Figure 1.31 with source node
1, sink node 11, 0 lower bound on all the arcs; capacity of 20 on all the horizontal arcs, 10 on all the vertical arcs, and 25 on all the diagonal

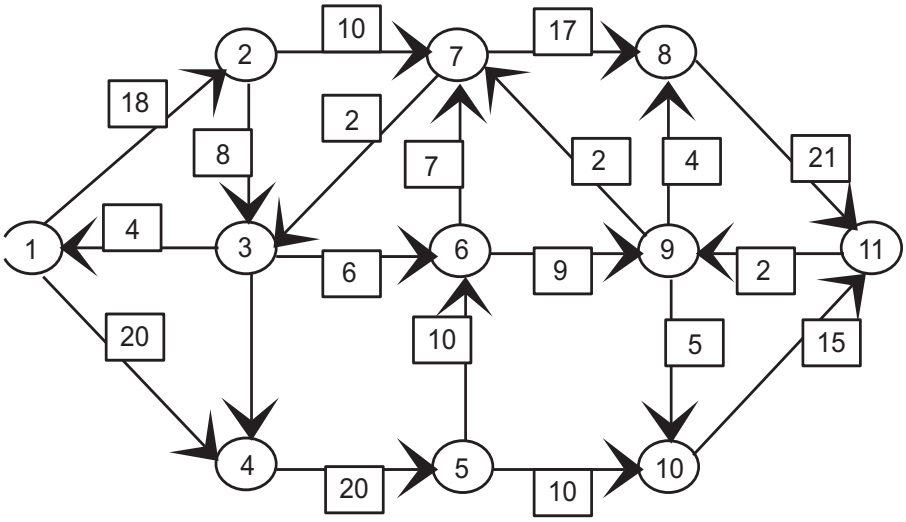


Figure 1.31:

arcs. A node-arc flow vector is entered in the boxes on the arcs. Check it for feasibility and find its value. Construct an arc-chain flow vector from it. What is its value? Explain.

1.34 If the arc-chain formulation of the maximum value flow problem in the directed single commodity flow network $G = (\mathcal{N}, \mathcal{A}, 0, k, \breve{s}, \breve{t})$ has an optimum solution, prove that it has one in which the arc-chain flow amounts are nonzero on at most m chains, where $m = |\mathcal{A}|$.

1.3 Formulation Examples and Applications

1.3.1 The Transportation Problem

A minimum cost flow problem on a directed network is called a transportation problem if: (1) every node is either a source node or a sink

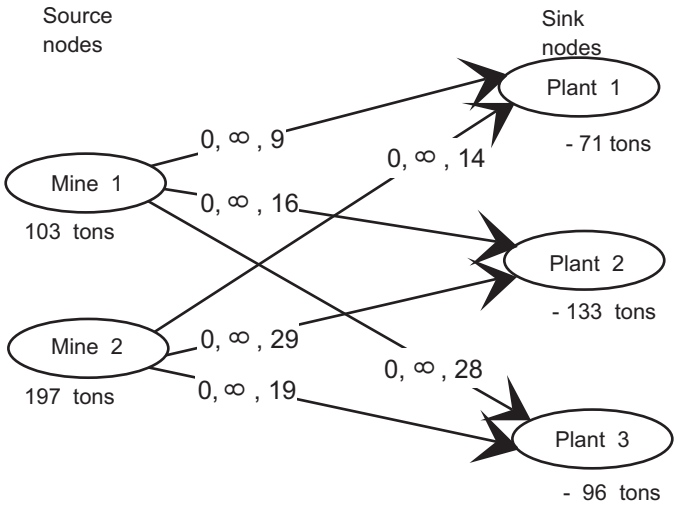


Figure 1.32: Bipartite network representation of the transportation problem. Data on each arc is its lower bound, capacity, and cost per unit flow. Exogenous flow amounts are entered by the side of the nodes.

node (so, there are no intermediate or transit nodes), and (2) every arc in the network joins a source node to a sink node. So it is a minimum cost flow problem on a bipartite network. We give an example of an uncapacitated (i.e., $k_{ij} = \infty$ for all the arcs) transportation problem. A steel company wants to minimize the total shipping bill for transporting iron ore from two mines to three steel mills subject to the data given below. A bipartite minimum cost flow model for it is shown in Figure 1.32.

	Shipping cost(cents/ton)			Availability
Plant $\rightarrow$	1	2	3	
Mine 1	9	16	28	103
2	14	29	19	197
Requirement	71	133	96	

1.3.2 The Assignment Problem

An assignment problem is a transportation problem in which the number of source nodes is equal to the number of sink nodes, and all the

	Expected annual sales in zone, if candidate assigned to zone.			
Zone →	1	2	3	4
Candidate 1	90	85	139	73
2	60	130	200	112
3	60	130	200	112
4	111	88	128	94

availabilities and requirements are equal to one. Here is an example: A large corporation is introducing a new product. There are 4 marketing zones, each requires a marketing director. Four candidates have been selected for these positions. Company estimates of the sales generated in the various zones are given in the table above, depending on which candidate is appointed in each zone. Assign candidates to zones to maximize total annual sales. This is a special bipartite minimum cost flow problem.

1.3.3 The Transshipment Problem

The transshipment problem is a minimum cost flow problem on a directed network which may not be bipartite, and in which there may be intermediate nodes, arcs joining a pair of sources, or a pair of sinks. Here is an example: A company manufacturing steel shelving cabinets has 3 plants, 2 packing units and 3 sales outlets. Plants 1 and 2 make shelves, and plant 3 makes the bars, screws and all the other components. Production of every item is measured in units of the item needed for one cabinet. Plant 1 has a production capacity of 20,000 cabinets/day, but the paint shop in it is small and can handle only 10,000 cabinets/day. Shelves made in plant 1 can be shipped in any quantity to plant 2 for painting as it has a very large paint shop. Plant 2 has a production capacity of 50,000 cabinets/day. The shelves move from plants 1 and 2 to plant 3, where they are combined with bars,

screws, etc. and shipped in bulk to either packing unit 1 or 2. Each packing unit combines the shelves, bars, screws etc. and packs them in cartons of one cabinet each. Shipping from the plants and packing unit 2 is by truck. The routes have the capacities (in cabinets/day) and costs (cents/cabinet) shown in Figure 1.33. Shipping from packing unit 1 to the retail outlets is carried out on water and is therefore cheaper.

Each packing unit can process at most 40,000 cabinets/day. The daily demand at sales outlets 1; 2; 3 is 25,00; 15,000; 22,000 cabinets respectively. We represent each plant, packing unit, sales outlet, by a separate node. The lines joining nodes represent the shipping channel between them. Data on each arc in Figure 1.33 is the lower bound (in cabinets/day shipped), capacity, and cost/cabinet, in that order. The problem is to determine an optimum production and shipping plan to meet the requirements at minimum shipping cost.

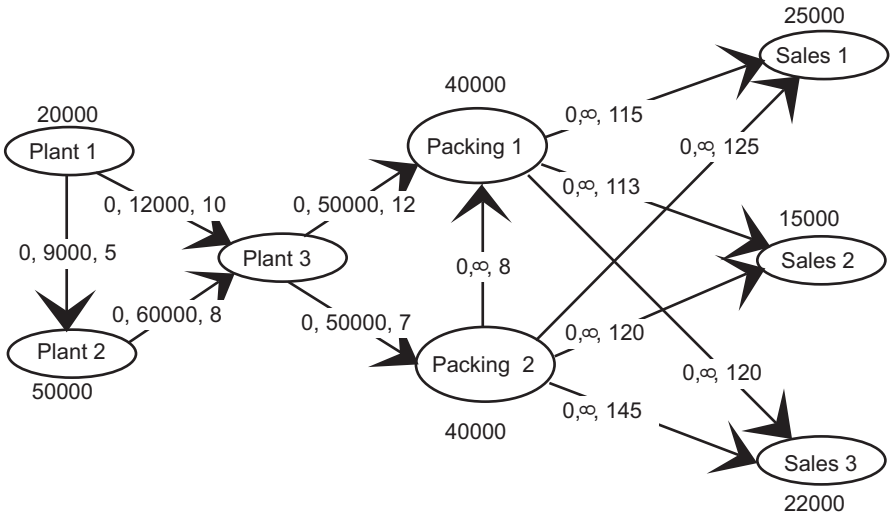


Figure 1.33: Numbers on nodes corresponding to plants and packing units are node capacities per day; and those on sales nodes are requirements per day.

Clearly, this is a minimum cost flow problem on a directed network which is not bipartite. The node corresponding to plant 2 is a source node, but it can also receive flow from other source nodes like plant 1. That's why problems like this are called **transshipment problems**.

In the same manner, problems involving production, in-process inventory, assembly, warehousing, or distribution can be represented as minimum cost flow problems on directed networks. In this model, the flow on each arc represents the amount of material (either finished or semi-finished) transferred from one manu- facturing unit to another. For such problems, clearly the node-arc flow model is the most direct. Since many network flow applications are of this type, the node-arc formulation is the one most commonly discussed.

1.3.4 An Application In Short Term Investments

A corporation receives money (income) from its dealers, and has expenses to be paid out (expenditures) every week. Expenses are light in the first half of the year, and tend to be high in the second half. Income, on the other hand, is normally high at the beginning and tends to level off towards the end. The planning horizon is 52 weeks (a year), and the corporation has surplus income at the beginning which it does not need for its expenditures until later. There are several securities in which the corporation can invest this surplus income on a short term basis for an integer number of weeks from 1 to 52. The yield from the pth security as a fraction of the amount invested is c_{pq}, if the maturity period is q weeks. The corporations expected income and expenditures for each week of the year are given, as also a lower and upper bound on the amount that can be invested in each security over the year by company policy. The problem is to determine an optimal short term investment plan to maximize the yield subject to the constraints. This is a problem of analyzing various cash flow alternatives and can be modeled as a network flow problem.

We provide a numerical example. To keep it small we consider a planning horizon of a year divided into four equal periods and only two securities, and we assume that all investments during a year have to be cashed that year itself. The yield from these investments is not treated as income for the company, but considered as if it were held in a separate account which is to be maximized. The expected income (in \$1000 units) in the four periods is 200, 180, 150, 170. The corresponding expenditures are 100, 110, 180, 250. The fractional returns

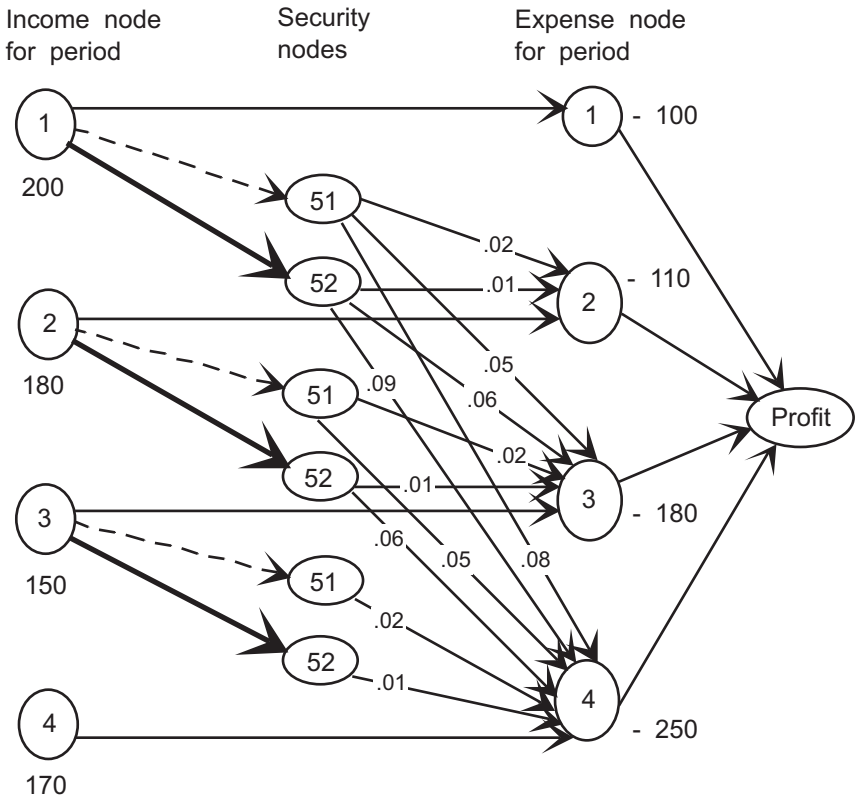


Figure 1.34:

from security 1 are 0.02, 0.05, and 0.08 if held for 1, 2, and 3 periods respectively. The corresponding figures from security 2 are 0.01, 0.06, 0.09. No more than 150 units can be invested in any one security over a year by company policy.

The network model is in Figure 1.34. It contains an income node and an expense node for each period, with the exogenous flow at them in money units entered by the side. Flow of money directly from an income to an expense node represents its use to meet expenses without being invested. Money flows through security nodes represent investments, and the data on the arcs joining a security node to an expense node represents the fractional yield on the flow through that arc. There is also a node called profit which accumulates the excess income of this

company over its expenses (remember that yield from investments is only counted in the objective function, and does not enter as a flow anywhere in this model).

In addition to the flow conservation equations, we have two other constraints. The sum of the flows on thick arcs has to be ≤ 150. Likewise, the sum of the flows on the dashed arcs has to be ≤ 150. These constraints are not of the network flow conservation type. So, this is a minimum cost network flow problem with additional linear constraints. An algorithm for such models is discussed in Chapter 6.

In constructing this model we treated the yield from investment as the objective function, and because of this it did not enter the flow at all. So flows across the investment arcs only consist of the principal amount, this guarantees that if a certain amount of money enters an arc (i, j) at node i, the same amount comes out at node j. Networks in which this property holds are called *pure networks*. All the problems we discussed so far involved only such networks. If the return on investment is also treated as income, then an amount of 100 units entering an investment arc (i, j) carrying 2% interest at node i, would become 102 units by the time it comes out at node j. Networks with this property are called *networks with gains or losses*, or *generalized networks*. An example of a generalized network flow problem is given in Section 1.3.7, and we present algorithms for solving them in Chapter 8.

1.3.5 Shortest Chain Problem

This is the problem of finding a shortest chain from an origin to a destination in a network $G = (\mathcal{N}, \mathcal{A}, c)$ with c as the vector of line lengths. This problem appears as a subproblem in many network applications. Shortest chain algorithms are discussed in Chapter 4.

1.3.6 Project Planning Problems

Large projects usually involve many individual jobs. There may be some interdependence among the jobs, some of them cannot possibly be started until others have been completed. This defines a precedence

ordering among the jobs, which can be represented by an acyclic network. Given the time needed for completing each job, the minimum time necessary to complete the project can be computed by solving a longest chain problem. Also, it may be possible to shorten the time needed to complete some jobs by spending extra money. In that case, the problem of determining how much of this extra money to spend on each job in order to complete the overall project within a specified duration at minimum extra cost can be posed as a minimum cost flow problem on an augmented network. This is the problem of computing the project cost curve as a function of project duration (see Chapter 7). This class of problems find many applications for network methods.

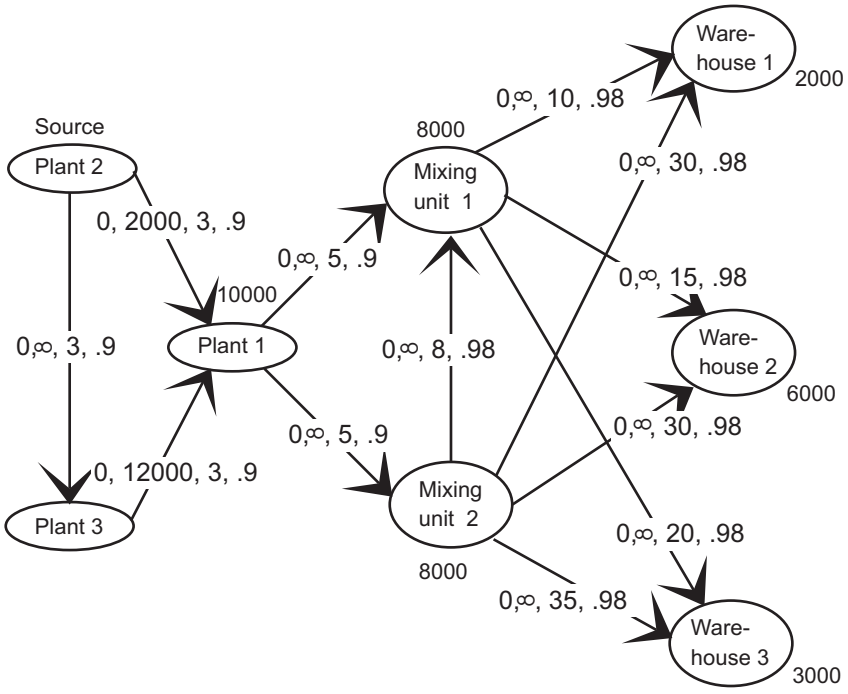


Figure 1.35:

1.3.7 Generalized Network Flow Problems

Consider a directed network in which flow entering an arc (i, j) at node i gets multiplied by a factor p_{ij} before it reaches node j. This may hap-

pen, for example, if losses occur during transit. If at least one of these multipliers is not equal to 1, such networks are called **generalized networks**, and flow problems on them are called **generalized network flow problems**. In contrast, the networks discussed so far are called **pure networks**, and the multiplier associated with every line is 1 in them.

Here is an example of a generalized network flow problem: A company manufactures 20-lb. bags of fertilizer containing N (nitrogen) and Ph (phosphorous) in fixed proportion. Their plant 1 manufactures urea containing N. Their plant 2, located near their phosphate mine, can only process a limited quantity of phosphate rock into a chemical compound containing Ph. The rest of phosphate rock is shipped to plant 3 for processing. Each plant ships its output to either of two mixing units where the chemicals are blended into fertilizer containing N and Ph in specified proportion and packed into 20-lb. bags. These bags are then shipped to 3 warehouses from which they are sold to retail outlets. From mixing unit 1, material is sent to warehouses by rail, but mixing unit 2 can only ship by truck. Rail transportation is cheaper, so sometimes fertilizer bags are sent from mixing unit 2 to mixing unit 1 for shipping by rail.

There is usually a 10% loss in transit due to spillage in the rock and chemicals shipped from the plants. There is also a loss of approximately 2% due to spoilage in the fertilizer bags shipped from mixing units to warehouses.

We measure each compound in units of it needed per 20-lb. bag of fertilizer. Plant 1 has a capacity of 10,000 bags/day. Plants 2 and 3 can process 2,000, and 12,000 bags/day respectively. Each mixing unit can process and ship 8,000 bags/day. It is assumed that the lower bound, capacity and shipping cost on arc (i, j) apply to f_{ij}, the flow entering this arc at its tail node i. The remaining data is given in Figure 1.35. Numbers by the side of Plant 1, and Mixing units 1, 2 are the associated node capacities. Numbers by the side of warehouse nodes are requirements. Data on each arc is its lower bound, capacity, cost per unit flow, and the multiplier, in that order.

The problem is to find a feasible flow vector that minimizes the shipping cost.

In the same manner, distribution problems in water, electric power, etc., where losses occur in transmission, can be modeled as generalized network flow problems. Algorithms for them are discussed in Chapter 8.

1.3.8 Applications In Routing

In this section, G denotes a connected undirected network $(\mathcal{N}, \mathcal{A}, c)$ where $\mathcal{N} = \{1, \ldots, n\}$, $\mathcal{A} = \{e_1, \ldots, e_m\}$, c_t is the length of $e_t, t = 1$ to m, and $c = (c_t) \geq 0$. For instance G might represent the street network of a town. There may be parallel edges in G. d_i denotes the degree of node i in G.

An **edge covering route (ECR)** or **postman's route** in G is an elementary cycle that begins at a node, travels along every edge at least once in some sequence, and returns to the starting node at the end. If an ECR passes l_t times through $e_t, t = 1$ to m, its length is $\sum_t l_t c_t$. The problem of finding a minimum length ECR is known as the **Chinese postman problem**.

An **Euler route** is an ECR that passes through each edge of the network exactly once. Since $c \geq 0$, if an Euler route exists in G, it must be a minimum length ECR.

THEOREM 1.11 *(Euler, 1736) There exists an Euler route in the connected undirected network* G *iff the degree of every node is even in it.*

Proof If Euler routes exist in G, select one and orient each edge in the direction in which it travels along that edge. An Euler route is a cycle. So, among those edges incident at a node, if the number with orientation leading into the node is r, then the number with orientation leading away from the node must also be equal to r. This implies that the degree of that node is $2r$, even. Hence, if an Euler route exists in G, the degree of every node must be even.

If every node in G has even degree, we discuss below an algorithm and prove that it will produce an Euler route in G. ∎

An undirected network is said to be an **Eulerian network** if it is connected and every node has an even degree in it.

Assume that G is Eulerian. A convenient way to represent an Euler route, is the **edge pairing representation** which we will describe now. Let Υ denote the Euler route $j_1, g_1, j_2, g_2, j_3, \ldots, g_m, j_{m+1} = j_1$ in G. So, $\{g_1, \ldots, g_m\}$ is a permutation of $\mathcal{A}$ and $g_t = (j_t; j_{t+1})$, for each $t = 1$ to m. In Υ, edge g_t is followed by g_{t+1}, hence we say that the edges (g_t, g_{t+1}) are paired in it, for $t = 1$ to m. Also g_1, g_m are the **first and last edges**. We indicate this by including $(0, g_1), (g_m, \infty)$ as pairs. These are called the **starting and finishing pairs**. This leads to the edge pairing representation for Υ as a list of ordered pairs

$$(0, g_1), (g_1, g_2), \ldots, (g_{m-1}, g_m), (g_m, \infty)$$

The number of pairs in this representation is $1 + |\mathcal{A}|$. The initial and final edges can be retrieved from the starting and finishing pairs. The initial node is the common node on these edges. The initial edge is travelled in the direction away from the initial node. If (g_t, g_{t+1}) is a pair in the representation, these edges must have a common node, say j; the route arrives at j by travelling through g_t and leaves through g_{t+1}. Each edge in G appears in exactly two pairs in an edge pairing representation, as the left hand member in one and the right hand member in the other. It is not necessary to record the various pairs in the representation in any particular order, but an order convenient to the driver is the order of travel.

As an example consider the Euler route in Figure 1.36 that begins at node 1, and travels through the edges in the orientation marked by their side. Its edge pairing representation is $(0, e_1), (e_1, e_2), (e_2, e_3), (e_3, e_4), (e_4, e_5), (e_5, e_6), (e_6, e_7), (e_7, \infty)$.

The **parity** of an integer is **even** if that integer is even, **odd** otherwise.

THEOREM 1.12 *Let $\mathbb{C}$ be an elementary cycle, and $\mathcal{P}_{ij}$ an elementary path between nodes i and j in* G*. The operation of deleting all the edges in $\mathbb{C}$ from* G*, or duplicating all the edges in $\mathbb{C}$ leaves the parity of the degree of every node unchanged. The operation of duplicating all the edges on the path $\mathcal{P}_{ij}$ changes the parity of the degrees of nodes i and j, but leaves that of all the other nodes unchanged.*

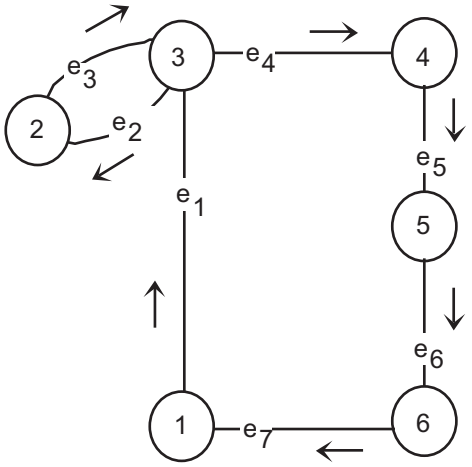


Figure 1.36: An Euler route with node 1 as the initial node.

Proof $\mathbb{C}$ contains an even number of edges incident at every node. $\mathcal{P}_{ij}$ contains an odd number of edges incident at i and j, but an even number at every other node. The results follow from these and the fact that the parity of an integer is unchanged by subtracting or adding an even number to it, but changes when an odd number is added to it. ∎

The algorithm for finding an Euler route in an Eulerian network G maintains a route which is always an elementary cycle, and grows it until it becomes an Euler route. At some stage, let $\hat{\mathrm{G}}$ be the network consisting of the edges in G not traversed in the present route. $\hat{\mathrm{G}}$ may not be connected, but each connected component in it is Eulerian by Theorem 1.12. Select a node i that is incident to some edges in the present route and to some not in it. Let (g_0, g_1) be an edge pair in the present route containing node i. Form an elementary cycle, $\mathbb{C}$ say, beginning and ending at i in $\hat{\mathrm{G}}$, and record it in edge pairing representation. Let h_0, h_1 be the first and last edges travelled on this cycle, so i is the common node on them. Delete the edge pair (g_0, g_1) from the route, and include into it the edge pairs $(g_0, h_0), (h_1, g_1)$ and all the edge pairs other than the starting and finishing pairs of $\mathbb{C}$. This has the effect of inserting $\mathbb{C}$ into the route. The new route follows the old one until the edge g_0 is traversed to reach node i, then follows $\mathbb{C}$

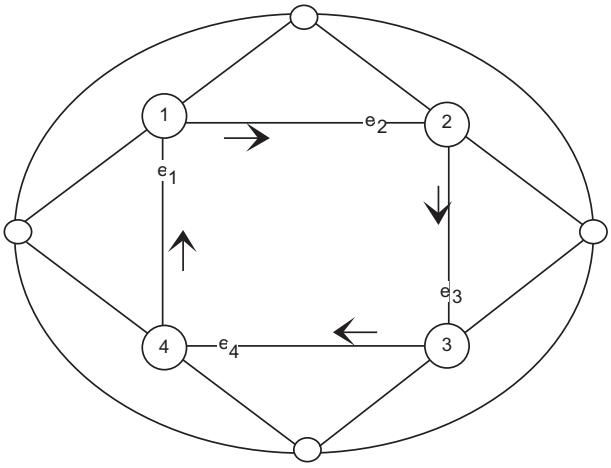


Figure 1.37:

until it is completed, and then follows the old route again. The new route contains more edges than the old. Repeat the same procedure until every edge is included in the route.

As an example, consider the Eulerian network in Figure 1.37. Start with the route $\{(0, e_1), (e_1, e_2), (e_2, e_3), (e_3, e_4), (e_4, \infty)\}$ consisting of the cycle $\mathbb{C}_1$ marked with arrows in Figure 1.37. When all the edges e_1 to e_4 traversed in the route are deleted from this network we get the network in Figure 1.38. This is connected, but in general such remaining networks may not be connected. We select node 3 on the route which is incident to some edges in the remaining network, and we find the cycle $\mathbb{C}_2 = \{(0, e_5), (e_5, e_6), (e_6, e_7), (e_7, \infty)\}$ beginning at node 3 in the remaining network, marked with arrows in Figure 1.38. Inserting $\mathbb{C}_2$ into the route leads to the new route $\{(0, e_1), (e_1, e_2), (e_2, e_5), (e_5, e_6), (e_6, e_7), (e_7, e_3), (e_3, e_4), (e_4, \infty)\}$. Now the edges e_5, e_6, e_7 on $\mathbb{C}_2$ are deleted from the remaining network in Figure 1.38, and the method is continued in the same manner.

During the algorithm, edges belong to two sets, the included (in the present route) and unincluded. The following symbols will be used; p denotes the initial node of the new elementary cycle being formed; g_0, g_1 denote either $0,\infty$, or the left and right edges in a pair in the present route incident at p; a denotes the first edge of the new elementary

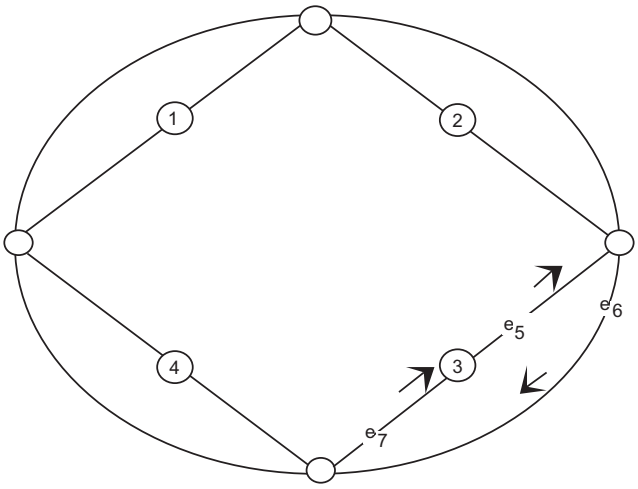


Figure 1.38:

cycle being formed incident at p; x denotes the present node on the new elementary cycle being formed; e denotes the unincluded edge incident at x selected for the new elementary cycle, and y the other node on it; and e_1 denotes an unincluded edge incident at y.

Algorithm for Finding an Euler Route

Step 1 Initialization Select any node, say 1, and an edge e_0 incident at 1. Set $g_0 = 0, g_1 = \infty, p = 1, x = 1, e = e_0, a = e_0$. All edges are unincluded.

Step 2 Select an unincluded edge Let y be the node $\neq x$ on e. Select an unincluded edge $\neq e$ incident at y, make it e_1 and go to Step 3. If none, we will have $y = p$. Go to Step 4.

Step 3 Make an edge pair Include the edge pair (e, e_1) in the list for the new cycle. Now e is included. Make e_1 into the new e, y into the new x, and go to Step 2 to select new y and e_1.

Step 4 Merging the cycle into the route Form the pairs (g_0, a) and (e, g_1) and insert these and all the other pairs generated in various occurrences of Step 3 in this cycle generation effort, into the route. Now e is included. Go to Step 5.

Step 5 Setup for new cycle If there are no unincluded edges, terminate. Otherwise, find a node incident to both included and unincluded edges, and make it the new p. Choose an edge pair containing p from the route, delete it from the list for the route, and make the right and left members in it the new g_0, g_1 respectively. Select an unincluded edge incident at p and make it the new e. Make $a = e, x = p$, and go to Step 2.

The reason for $y = p$ when no unincluded edges are found incident at y in Step 2 is the following. At this stage $e = (x; y)$ and all edges other than it incident at y are included. Suppose $y \neq p$. What we have traced so far is a path from p to y. All the edges on this path from p up to x are 'included'. The number of unincluded edges incident at y is an even integer, and the present edge $e = (x; y)$ is one of them, hence there must be at least one more unincluded edge incident at y, a contradiction to our hypothesis. Hence when the algorithm arrives at Step 4 we must have $y = p$, and the elementary cycle being traced must be complete, and e is the last edge on it.

Each time Step 4 is completed, a new elementary cycle is formed and inserted into the route, and the set of remaining (i.e., unincluded) edges forms one or more Eulerian networks. Each time Step 3 or 4 is carried out, one new edge is 'included', so together they occur $m = |\mathcal{A}|$ times in the algorithm. If $|\mathcal{N}| = n$, Step 2 takes at most $O(n)$ effort, and it is automatically followed by Step 3 or 4. So, with an effort of at most $O(nm)$, the algorithm is guaranteed to find an Euler route in G.

Comment 1.1 This algorithm for finding an Euler route in an Eulerian network is due to J. Edmonds and E.L. Johnson [1973]. Their paper discusses other ways of representing Euler routes, and several other algorithms for finding Euler routes.

Now consider the case where G is connected, but not Eulerian. So, the total number of odd degree nodes in G is a positive even number. Suppose these are $1,\ldots,2p$. Let Φ be an ECR that passes l_t times through edge $e_t, t = 1$ to m. Obtain the network G_Φ by copying the edge e_t exactly l_t times, for $t = 1$ to m. Then Φ must be an Euler route in G_Φ, and hence every node in G_Φ must be an even degree node. For

$t = 1$ to m, define $b_t = 1$ if l_t is odd, 2 otherwise. Let $\tilde{\mathrm{G}}$ be the network in which the edge set contains exactly b_t copies of e_t for $t = 1$ to m. Since $l_t - b_t \geq 0$ and even for each t, $\tilde{\mathrm{G}}$ is Eulerian too. Let $\tilde{\Phi}$ be an Euler route in $\tilde{\mathrm{G}}$. The length of $\tilde{\Phi}$ is $\sum_{t=1}^{m} b_t c_t \leq \sum_{t=1}^{m} l_t c_t =$ length of Φ. This clearly implies that there exists an optimum (i.e., minimum length) ECR in G which passes through each edge of G at most twice. Hence we consider only such ECRs in the sequel. A minimum length ECR of this type must minimize the sum of lengths of edges in $\bar{\mathcal{A}}$, the subset of edges that it passes through twice.

Every odd degree node in G becomes an even degree node, and every even degree node remains an even degree node when edges in the repeated set $\bar{\mathcal{A}}$ are duplicated in G. By Theorem 1.12, this implies that the odd degree nodes $1, \ldots, 2p$ can be partitioned into p pairs, say $(i_{11}, i_{12}), \ldots, (i_{p1}, i_{p2})$, such that there is a path from i_{r1} to i_{r2} among the set of edges $\bar{\mathcal{A}}$, and $\bar{\mathcal{A}}$ is the set of edges on these paths. As an example, consider the ECR 1,e_1, 4, e_6, 3, e_5, 2, e_3, 1, e_2, 3, e_2, 1, e_1, 4, e_4, 2, e_3, 1, in the network in Figure 1.39. In this ECR, the thick edges $\{e_1, e_2, e_3\}$ have been traversed twice. This set is the union of two paths 1,e_2, 3 and 2,e_3, 1, e_1, 4 between the pairs of odd degree nodes 1, 3 and 2, 4 respectively. In order to minimize the sum of the

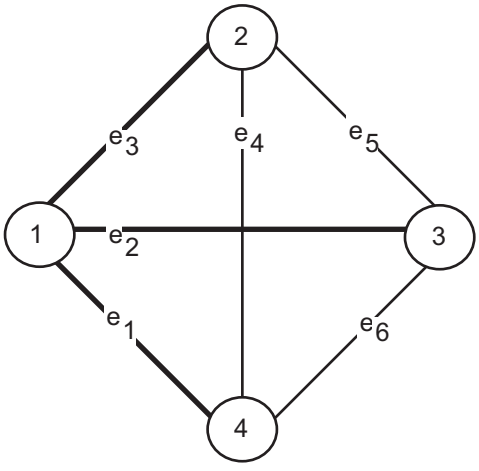


Figure 1.39: The thick edges are those traversed twice by the ECR under consideration.

lengths of the edges traversed twice, the path which has been duplicated between a pair of odd degree nodes should be a shortest path. Also the partitioning of the odd degree nodes in G into pairs should be done in such a way that the sum of the lengths of the shortest paths between the various pairs in the partition is minimized. Hence an optimum ECR in G can be obtained by the following procedure.

1. Find the shortest paths between all pairs of odd degree nodes in G. Let $\mathcal{P}(i,j)$ be a shortest path of length d_{ij}, for $i \neq j = 1$, to $2p$. Let $D = (d_{ij})$ be the $2p \times 2p$ shortest path distance matrix with $d_{ii} = \infty$ for all i.

2. Let $\hat{\mathcal{N}} = \{1, \ldots, 2p\}, \hat{\mathcal{A}} = \{(i,j) : i,j \in \hat{\mathcal{N}}, i \neq j\}, \mathrm{H} = (\hat{\mathcal{N}}, \hat{\mathcal{A}})$. H is the complete undirected network on the set of odd degree nodes in G. The arc $(i,j) \in \hat{\mathcal{A}}$ corresponds to the shortest path $\mathcal{P}(i,j)$ between the odd degree nodes i,j in G, and we define its weight to be d_{ij}. Find a minimum weight perfect matching in H. The blossom algorithm discussed in Chapter 10 can be used for this.

3. If $\{(i_{11}; i_{12}), \ldots, (i_{p1}; i_{p2})\}$ is a minimum weight perfect matching in H, an optimum pairing for the odd degree nodes in G is (i_{r1}, i_{r2}), $r = 1$ to p. Duplicate all the edges along the shortest paths $\mathcal{P}(i_{r1}, i_{r2}), r = 1$ to p in G, obtaining an Eulerian network, $\bar{\mathrm{G}}$. Find an Euler route in $\bar{\mathrm{G}}$. It is an optimum ECR in G.

The chinese postman problem provides an important application for the blossom algorithms discussed in Chapter 10. Many routing problems in trash collection, road sweeping, school bus route planning etc. can be modeled using the chinese postman problem.

1.4 Exercises

1.35 Let $\mathrm{G} = (\mathcal{N}, \mathcal{A})$ be a non-Eulerian undirected connected network with c as the vector of edge lengths. A subset of edges $\mathbf{F}$ in G is said to be a *feasible subset* if their deletion from G makes each connected component in the remaining network Eulerian. Discuss an efficient algorithm for finding a minimum length feasible subset. (S. Biswas)

1.36 Prove that every doubly stochastic matrix, i.e., a feasible solution of (1.15), can be expressed as a convex combination of assignments.

1.37 Capacity Acquisition Problem A company requires a minimum of d_h units of warehouse capacity in period $h = 1$ to n. Capacity acquired at the beginning of period h and relinquished at the beginning of period t is said to be acquired for the interval $[h, t]$. This costs c_{ht} per unit. Let x_{ht} denote the units of capacity acquired for the interval $[h, t]$. Let y_j denote the units of acquired but unused capacity during period $j = 1$ to n. Formulate the problem of finding a minimum cost capacity acquisition to meet the requirements as an LP. Show that this problem can be transformed into a network flow problem by simple linear transformations. Do it for $n = 4$, and draw the corresponding network. (Veinott and Wagner [1962])

1.38 A company making a single product has a production capacity of 25,000 tons per period. They have to ship out respectively 15,000, 18,000, 30,000, and 8,000 tons in periods 1 to 4. The expected production cost ($/ton) during periods 1 to 4 is 50, 60, 40, and 70 respectively. The production during a period can either be shipped out in the same period or stored for later shipment at a storage cost of $2/ton/period, the charge being imposed on the quantity in storage at the end of the period. Initial inventory is zero; final inventory at the end of period 4 should be zero too. Formulate the problem of determining an optimum production storage plan as a transportation problem.

1.39 Application in Marketing Here is a multibrand, multiattribute marketing model. Marketing is replete with examples where two brands may have the same product attribute values but enjoy very different market shares, so we include an additional component called 'brand specific effect' which measures the overall preference not explained by the attributes used in the model. This may depend on the levels or the strategy of the brand's marketing effort, etc. Let $j = 1$ to n be the different brands, $i = 1$ to m a representative sample of consumers, $p = 1$ to t the relevant product attributes, y_{jp} = brand j's value on the pth attribute, w_{ip} = estimated importance weight of the ith customer to the pth attribute, $b_{ij} = \sum_{p=1}^{t} w_{ip} y_{jp}$ = preference

measure of customer i for brand j, v_j = brand specific effect of brand j, and c_i the brand chosen by the ith consumer.

	b_{ij}		
$j \rightarrow$	1	2	3
$i = 1$	9	8	12
2	13	11	15
3	18	13	15
4	16	10	4
5	19	5	3
6	8	6	7

Given the v_j, $b'_{ij} = b_{ij} + v_j$ defines the overall preference of consumer i to brand j. It is reasonable to assume that consumer i would choose that brand j with the largest b'_{ij}, and this choice is not altered by adding the same additive constant to all the v_j. Thus, the v_js can only be determined up to an additive constant. Given the v_j, define $s_i = \max \{b'_{ij} : j = 1 \text{ to } n\}$. The problem is to determine the v_j to get the best fit. Clearly, this requires minimizing $\sum_{i=1}^{m}(s_i - b'_{i,c_i})$. Show that this can be done using algorithms for the transportation problem.

Consider the numerical problem in which $n = 3$, $m = 6$, $c = (c_i) = (1, 1, 2, 2, 2, 3)$ and b_{ij}s are tabulated above. Obtain the best estimates for (v_1, v_2, v_3) from this data.
(Srinivasan [1979]).

1.40 Mold Allocation Problem in a Tire Plant To make a tire, one has to set up a "mold" into a general purpose machine called a "cavity" which is carried out by highly skilled personnel with special equipment. Every cavity can produce any type of tire, given the appropriate mold. Molds are very expensive, as they take 6 months to prepare, and they usually outlast the product for which they are designed, and hence are normally converted to another type. Assume that setups are carried out only at the beginning of each period, and consider the problem of determining the assignment of molds to cavities. The following data is given: n = number of periods in the planning

horizon, m = number of mold types, C = number of cavities available, T_{it} = number of type i molds available during period t, k_t = max. possible number of setups at the beginning of period t, l_{it} = lower bound on type i molds in cavities period t, and α_{it} = cost (\$) of setup of a type i mold in a cavity at the beginning of period t. x_{io} are predetermined nonnegative integer constants that give the initial distribution of molds in cavities. The decision variables (nonnegative integer variables) are: x_{it} = number of type i molds in cavities in period t, y_{it} = number of setups of type i molds performed at the beginning of period t, and z_{it} = number of takedowns (removal of molds from cavities) of type i molds performed at the beginning of period t.

Formulate the problem of determining the optimum values of x_{it}, y_{it}, z_{it} subject to the stated constraints as a linear integer program. Develop a transformation that permits the reduction of this problem into a minimum cost flow problem. Carry out this transformation in the numerical problem with $m = 2$ and $n = 3$ $C = 15, x_{10} = 6, x_{20} = 5$, and the following data.

	T_{it}		l_{it}		α_{it}		k_t
	$i=1$	2	$i=1$	2	$i=1$	2	
period 1	7	7	5	6	100	150	3
2	7	9	5	8	100	175	4
3	6	11	4	10	90	140	5

(Love and Vemuganti [1978])

1.41 Sometimes an activity can only be carried out if other activities are also carried out. Here is the data on such a situation in which there are 7 projects each of which can either be completely carried out or not at all, and only if other specified projects are also carried out. Formulate the problem of determining which subset of projects should be carried out to maximize the total net profit, as a 0-1 integer program, and show how it can be solved using a network flow approach.

Project No.	Net Return from project (million $)	Project can be carried out only if these other projects are also carried out
1	10	2
2	−8	
3	2	1,5
4	4	2,6
5	−5	
6	3	
7	2	3

(Williams [1982], and Baker [1984])

1.42 Allocating Oil Wells to Platforms The variable p is the number of platforms to be built, and w is the total number of production wells to be drilled in an oilfield, all with known locations. Platforms have to be built first, the size and cost of each depends on the number of wells to be drilled from it, and its location. The decision variable t_{ij} is 1 if production well i is assigned to platform j, 0 otherwise, for $i = 1$ to $w, j = 1$ to p, and $m_j = \sum_{i=1}^{w} t_{ij}$ is the number of production wells assigned to platform j. We are given $g_j(m_j) =$ cost of building platform j as a function of m_j, a piecewise linear convex function, and $c_{ij} =$ cost to drill well i from platform j once it is built, $i = 1$ to w, and $j = 1$ to p. A minimum cost allocation is obtained from the following problem. Show that this problem can be transformed into a minimum cost pure network flow problem.

$$\text{minimize} \quad \sum_{i=1}^{w}\sum_{j=1}^{p} c_{ij}t_{ij} + \sum_{j=1}^{p} g_j\left(\sum_{i=1}^{w} t_{ij}\right)$$

$$\text{subject to} \quad \sum_{j=1}^{p} t_{ij} = 1, i = 1 \text{ to } w$$
$$t_{ij} = 0 \text{ or } 1 \text{ for all } i, j.$$

(Divine and Lesso [1972])

1.43 A department in a university has admitted n students in a term. They have a_1 graduate assistantships (GA's, full tuition + stipend)

and a_2 tuition fellowships (TF's, tuition only) to offer. The variables p_i^1, p_i^2, p_i^3 denote the probabilities that the ith student accepts the admission if he is awarded GA, TF, or is not awarded any of these, respectively, estimated by the admissions office based on their background. W_i denotes a desirability rating given by the department to the ith student for $i = 1$ to n. Formulate the problem of selecting the set of students to be offered GA's and TF's, so as to maximize the total expected desirability rating of the incoming batch (assume that if a student is offered some aid and does not accept, then he is lost, and this aid cannot be offered to someone else). (Chandrasekaran and Subba Rao [1977])

1.44 Chromosome Classification Karyotying is a process by which chromosomes are classified into groups by observing features like size, shape, band structure induced by staining, etc. Suppose there are n chromosomes to be assigned to m groups, and it is known that the jth group has b_j chromosomes. After observations, it has been estimated that the probability of the ith chromosome belonging to the jth group is p_{ij}, these p_{ij} are given. It is required to assign chromosomes to groups so as to maximize the product of posterior probabilities subject to achieving correct group totals. Formulate this as a transportation problem.
(Tso [1986])

1.45 The Single Depot, Unconstrained Number of Vehicles Bus Scheduling Problem There are n trips to be operated in a planning interval T, each characterized by its starting and ending time, starting and ending places, and by its line (company may operate several lines, and typically incurs a penalty p_1\$ whenever trips of two lines are combined into a bus schedule). Assume that trips are ordered $1, \ldots, n$, by increasing value of starting time. Trip j can follow trip i in a bus schedule only if the starting time for trip j exceeds the ending time for trip i plus the driving time from ending place of i to starting place j computed with a fixed safety margin. All such pairs are specified, as well as the dead-heading cost, q_{ij}, from ending place of i to starting place of j for each such pair (i, j). Also, D, the cost incurred by each bus used in the schedule is given. Formulate the

problem of forming minimum cost bus schedules as an assignment or transportation problem. Discuss how this formulation changes if there is a bound on the maximum number of vehicles to be used. (Gavish and Schweitzer [1974], Gavish, Schweitzer and Shlifer [1978], Pinto Paixão and Branco [1987], Bertossi, Carraresi and Gallo [1987])

1.46 The Caterer Problem A caterer has to supply clean napkins each day over a period of n days. Soiled napkins can be laundered by a slow process that takes p days at a cost of $r \geqq 0$ per napkin, or by a fast process that takes $0 < q < p$ days and costs $d > r$ per napkin. Also new napkins can be bought, each at a cost of $b > d$ any day. The demand, given to be a_i napkins on the ith day, $i = 1$ to n, is to be met at least cost. For the first q periods, napkins must be purchased since soiled napkins cannot be laundered quickly enough for reuse. So, initially $a_1 + \ldots + a_q$ purchased napkins are needed. Denote the total number of new napkins purchased by $a_0 + a_1 + \ldots + a_q$ where $a_0 \geqq 0$ is treated as a parameter. Prove that a feasible solution exists iff $a_{\min} \leqq a_0 \leqq a_{\max}$, where

$$a_{\min} = \max\left\{0; \sum_{j=q+1}^{q+h} a_j - \sum_{i=1}^{h} a_i, \quad \text{for} \quad h = 1, 2, \ldots, n-q\right\}$$

$$a_{\max} = \sum_{j=q+1}^{n} a_j.$$

For given a_0 satisfying these feasibility conditions, show that the problem can be formulated as a $(1 + n - q) \times (n - q + 1)$ balanced transportation problem with rows of the array corresponding to 0, day $1, \ldots,$ day $n - q$, and columns corresponding to day $q + 1, \ldots,$ day n, slack; and variables x_{0j} = number of purchased napkins used on day j, x_{ij} = number of napkins soiled on day i and reused on day j, for $i \neq 0, j \neq n + 1$, $x_{i,n+1}$ = slack variable. Develop a special direct method of $O(n)$ computational effort to find an optimum solution of this transportation problem, based on the concept that slow laundered napkins are reused at the earliest possible moment, while fast laundering is delayed as much as possible. Solve the numerical problem

corresponding to data, $n = 10, q = 2, p = 5, r = 2, d = 4, b = 10, (a_0$ to $a_{10}) = (3, 7, 12, 2, 6, 9, 6, 13, 8, 14, 6)$.

Develop a special direct method to obtain optimum solutions for all integer values of the parameter a_0 in its feasibility range, and for finding the best value for a_0. Apply this method to find the optimum a_0 for the numerical problem given above. (Szwarc and Posner [1985]).

1.47 Application in School Planning The variable m is the number of school districts in a region, which has n schools in the public school system. For $i = 1$ to m, a_i is the number of students who will attend the public schools from district i. For $j = 1$ to n, b_j is the maximum number of students that school j can accommodate (this is typically the number of classrooms multiplied by the maximum number of students allowed per class, which is normally set at 25 or so.) For $i = 1$ to $m, j = 1$ to n, c_{ij} is the distance between district i and school j (this is usually the bird's flight distance between the school and the demographic center of gravity of the district). It is required to determine the number of students in each district to be assigned to each school so that no school is filled beyond its capacity, every student gets assigned to a school, and the total distance between home and school for all students is minimized (it is OK to split a district between schools, as the districts could be subdivided and renumbered). Formulate this problem.

1.48 There are m insurance agents in a region divided into n small localities called blocks. We are given the following information: $w_j =$ expected workload (man-hours per year) in block $j, j = 1$ to n; $d_{ij} =$ distance of block j to the location of agent $i = 1$ to m, $j = 1$ to n; a_i = ideal fraction of workload in region to be assigned to agent $i (a_i > 0$ for all i and $\sum a_i = 1)$; $a_i(1 - f_i), a_i(1 + f_i) =$ lower, upper bounds on fraction of region's workload to be assigned to agent $i (0 < f_i < 1)$.

Let $x_{ij} =$ amount of workload in the block j assigned to agent i and assume that travel cost of agent i to block j is $x_{ij}d_{ij}$. Give a network flow formulation for assigning workloads to the agents, to minimize the travel costs of all the agents put together. (Marlin [1981])

1.49 There are r distinct groups of people planning to vacation on the beach together one night. The ith group has n_i people in it, $i = 1$

to r. There are p cars available for the drive, the jth car can seat d_j people, for $j = 1$ to p. It is required to find a seating arrangement so that no two members of the same group are in the same car. Formulate this as a network flow problem.

1.50 Natural Gas Distribution The gas pipeline network consists of three supply systems, 1, 2 and 3. Supply system 1 consists of two source nodes with supplies of 500 ft^3 of gas each. Supply system 2 has a single source node with a supply of 1,000 ft^3, and supply system 3 has two source nodes with supplies of 2,000 and 6,000 ft^3. In each supply system all the gas flows out through a transfer node to which each source node in that system is connected by a pipeline. There are two natural gas users in the network. The delivery to user 1 has to be between 3,000 to 3,500 ft^3, and the delivery to user 2 has to be between 6,000 to 7,000 ft^3. User 1 is connected by a direct pipeline to supply systems 1 and 2, and user 2 is connected likewise to supply systems 1 and 3. There is also a redistribution node which is connected by pipeline to each supply system and to each user. Every pipeline through the redistribution node has a capacity of 3,000 ft^3. All other pipelines in the network have a capacity of 2,000 ft^3. Formulate the problem of minimizing the total flow through the redistribution node while meeting the delivery obligations.

1.51 Allocation of Contractors to Public Works A region is geographically divided into r districts. In each district there is public work to be carried out which has to be contracted out. There are s_1 experienced and s_2 inexperienced contractors available. The work is actually carried out by teams provided by contractors and sent to the districts for this purpose. For $j = 1$ to $s_1 + s_2, n_j$ is the maximum number of teams that the jth contractor can provide. For $i = 1$ to r and $j = 1$ to $s_1 + s_2, c_{ij}$ is the price quoted by the jth contractor to send one team to district i for doing the work there. N_i is the minimum number of contractors to be allocated to district i. By policy, each district must get at least one team from an experienced contractor. Give a network formulation for the problem of allocating teams from the contractors to the districts, subject to the above constraints, at minimum cost. Construct this network formulation for the following

data (c_{ij} are in units of $\$10,000$): $r = 5, s_1 = 2, s_2 = 4, n = (n_j) = (3, 4, 6, 8, 10, 5), N = (N_i) = (2, 3, 4, 2, 3)$.

$$c = (c_{ij}) = \begin{pmatrix} 35 & 48 & 21 & 33 & 41 & 28 \\ 56 & 29 & 19 & 22 & 38 & 50 \\ 45 & 48 & 43 & 41 & 46 & 43 \\ 65 & 58 & 54 & 59 & 52 & 51 \\ 76 & 81 & 79 & 80 & 69 & 68 \end{pmatrix}$$

(Cheshire, McKinnon, and Williams [1984])

1.52 Budget Allocation There is a four-level hierarchy in the allocation of a state's educational budget. At the top is the state with its educational budget. The next level consists of the various universities, or campuses which are separate budget entities. The next level corresponds to colleges within each university. Finally, the lowest level corresponds to the departments within each college. Formulate this as a network flow problem, where the flow represents the budget allocation to the various units in each level. Each arc in the model will have lower and upper bounds, where lower bound = minimal requirements, and upper bound = budget request submitted by the unit administrator. The total budget is constrained by availability of state funds for education. An objective function which is composed of a weighted average of the allocations is to be optimized. The weights represent the relative importance given to the unit by the decision makers (for example, the weights may be proportional to the corresponding enrollment projections).

Discuss the changes to be made in the formulation for determining the annual budget allocations in a multiyear planning horizon, if transfers are allowed from one year to the next.

1.53 Industrial Estate Development A country is making a 15-year plan for industrial land development. There are 17 sites where land is available to be developed. The parameter a_i denotes the maximum amount of land (acres) available for industrial development at site i; c_i (in thousand \$/acre) is the present cost (at the beginning of the planning horizon) of developing industrial land at site i; and r_i (in

thousand \$/acre) is the discounted revenue collectable over the lease period for an acre of industrial land leased out at site i. This data is tabulated below.

i	1	2	3	4	5	6	7	8
c_i	130	130	35	31	31	18	87	26
r_i	28	28	45	45	28	28	28	28
a_i	250	350	32	532	350	60	74	30

i	9	10	11	12	13	14	15	16
c_i	17	23	30	22	31	131	65	22
r_i	28	28	28	102	113	142	85	113
a_i	45	30	102	25	164	1593	321	2133

Sites 1 to 5 are considered highest priority sites. At these sites there is a requirement that a minimum of 40, 50, 30, 50, and 50 acres must be developed and leased out in the first 10 years of the planning horizon. Sites 6 to 11 are at the next priority level. At these sites there is a requirement that a minimum of 30 acres in each site must be developed and leased out during the planning horizon. The projected demand for industrial land development at all sites put together in year t of the planning horizon is d_t acres where $d_t = 300, 300, 300, 350, 350, 400, 400, 400, 400, 400, 450, 450, 450, 450$, and 450, for $t = 1$ to 15 respectively. Formulate the problem of allocating land at the various sites for industrial development, over the years of the planning horizon, subject to the constraints mentioned above, so as to minimize the total net discounted cost (cost of developing minus the discounted revenue collected), as a minimum cost network flow problem. (Fong [1980])

1.54 A Minimum Cost Supply-Demand Problem Over an $n-$period planning horizon a business person can buy, sell, or hold the commodity for later sale, subject to the following constraints. In the

ith period, $k_i \geqq 0$ is an upper bound on the amount of commodity he can buy, $d_i \geqq 0$ is an upper bound on the amount of commodity he can hold till next period, and $\ell_i \geqq 0$ is a lower bound (because of commitments made already) on the amount he sells. The buying, selling and storage costs are $a_i \geqq 0, b_i \geqq 0, c_i \geqq 0$ respectively in the ith period. It is required to determine his optimum buying, selling, holding plan over the planning horizon, in order to maximize the net total profit. Formulate this as a minimum cost flow problem on an acyclic network. (Ford and Fulkerson [1962])

1.55 A Transshipment Model for Leveling a Road Bed When building a road through mountainous terrain, earth has to be redistributed from high points to low points to produce a relatively level road bed. The engineer must determine the number of truckloads of earth to move between various locations along the proposed road for leveling the route. Thus, high points along the proposed road bed are viewed as sources of earth, while the low points are correspondingly sinks. A terrain graph is an undirected network with nodes on it representing locations at which there are deficits (negative exogenous flow w_i) or surpluses (positive exogenous flow w_i), and edges on it representing the available routes for redistribution of earth, with the cost coefficient $c(e)$ of edge e representing the traversal cost of that edge. $c(e) > 0$ and $\sum w_i = 0$. A leveling plan is a nonnegative flow vector in this network that fulfills the requirements at all the nodes. Formulate the problem of finding a minimum cost levelling plan as a transshipment problem. Construct this model for the road construction situation described in Figure 1.40
(Farley [1980])

1.56 Machine Loading Problem There are m products to be produced on n machines. Each product can be produced on any machine, but it takes p_{ij} units of machine time and costs C_{ij}\$ to produce one unit of product $i = 1$ to m on machine $j = 1$ to n. At most b_j units of machine time is available on machine $j = 1$ to n, and it is required to produce a_i units item $i = 1$ to m during a period. Formulate the problem of finding an optimum production plan as a network flow problem. (Iri, Amari and Takata [1968])

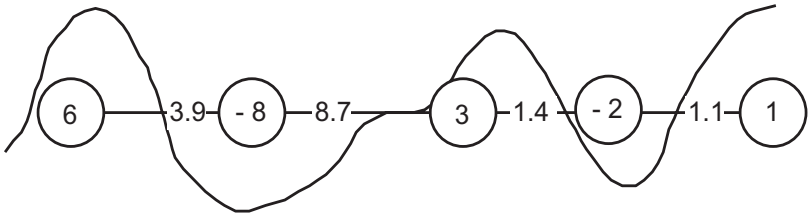


Figure 1.40: Locations on road bed are marked with circles, with the exogenous flow amount at that location entered inside the circle. Lengths of edges are marked on them.

1.57 G = $(\mathcal{N}, \mathcal{A})$ is a given directed network. For each $i \in \mathcal{N}$, we are given nonnegative integers a_i', a_i'' and b_i', b_i'' satisfying $a_i' \leqq a_i''$ and $b_i', \leqq b_i''$. It is required to find a subnetwork $\overline{\mathrm{G}}$ of G, satisfying the property that for each $i \in \mathcal{N}$ the indegree of i in $\overline{\mathrm{G}}$ is between a_i' and a_i'', and the outdegree of i in between b_i' and b_i''. Formulate this as a flow problem.

Suppose we are also given a vector c of arc cost coefficients in G. Define the cost of a subnetwork to be the sum of the cost coefficients of arcs in it. Discuss a formulation for the problem of finding a minimum cost subnetwork of G subject to the in- and outdegree constraints at the nodes described above, as a flow problem.

Note 1.1 The corresponding degree constrained subnetwork problem in undirected networks cannot be reduced to a max-flow problem, but it can be posed as a matching problem. See Exercise 10.19.

1.58 A medical college has 57 students to be assigned to internships at facilities over a period of 3 terms. There are 3 classes of facilities, 1) speciality, 2) rehabilitation, and 3) general/acute. Each student must intern for one term in a facility of each class. Each facility specifies the maximum number of interns it can take each term. Each student is allowed to specify three choices for each class of facility and when (which of three terms) he or she desires to intern there. The problem is to assign the students to the facilities for internship in a manner that maximizes the number of students receiving their preferred facility in

the term they asked for it. Develop an efficient approach for solving this problem.

1.59 Airline Fueling Problem Consider an $n-$leg flight for an airline with S_1 as the origin, S_{n+1} as the destination, and the ith leg consisting of a nonstop flight from S_i to $S_{i+1}, i = 1$ to n. We have the following data: d_i = normal fuel requirement (tons) for ith leg; t_i = fuel capacity (tons) on this leg based on known load; s_i = maximum fuel available (tons) for this plane at S_i; c_i = \$/ton of fuel at S_i; g_i = fuel remaining when S_{i+1} is reached per ton of extra fuel over d_i carried at S_i. Denoting by y_i the tons of fuel purchased at S_i; and by x_i the excess fuel in the plane tanks at the point of take-off from S_i, $i = 1$ to n, formulate the problem of minimizing the cost of fuel for this entire flight, as a generalized network flow problem. (Queyranne [1982])

1.60 Steel Slab Cutting With A Flame Torch On a rectangular steel slab a cutting pattern is laid out which is a planar multinetwork. It is required to execute this cutting pattern using a flame torch. Cutting always begins at a node. To begin, the torch has to be held at that node for some time until the flame reaches from the top to the bottom (called a blowthrough) and then the torch can be moved easily along any simple path until it is lifted again. The torch cannot travel an edge a second time. If it is necessary to return to a node after passing through it once, it is necessary to make another blowthrough at it. Discuss an algorithm for determining the flame torch route to minimize the number of blowthroughs needed. What is the optimum objective value? (Manber and Israni [1984])

1.61 There are n students in a projects course. There are m available projects, with project i having a capacity of b_i students, $i = 1$ to m. Each student has to work on precisely one project. If there are no students to work on a project, it can be dropped. A total of r supervisors are available. Corresponding to each project a subset of one or more supervisors who can supervise students working on that project is specified. The parameter k_p is the maximum number of students that supervisor p can handle, $p = 1$ to r. Each student specifies a subset of projects arranged in descending order of preference. The objective

is to assign students to projects and supervisors so that each student gets to work on a project that has his most preferred ranking, as far as possible. Taking as an objective function the sum of the rankings of the projects the students work on, formulate the problem of doing these assignments as a minimum cost flow problem.

1.62 In a tournament there are n players. Every pair of players play against each other precisely once, and the rules of the game exclude draws. A vector of nonnegative integers $(s_1, \ldots, s_n)$ is called a score vector for this tournament, if s_i, is the number of wins recorded by the ith player in this tournament, for $i = 1$ to n. Given a nonnegative integer vector $b = (b_1, \ldots, b_n)$, it is required to check whether b can be the score vector in such a tournament. Formulate this as the problem of finding a feasible flow vector in a capacitated bipartite network.

1.63 G $=(\mathcal{N}, \mathcal{A}, 0, k, V)$ is a directed connected single commodity flow network with V as the vector of exogenous flow amounts at the nodes. For each $(i, j) \in \mathcal{A}, r_{ij}\$$ is the cost of augmenting the flow capacity on arc (i, j) by 1 unit. It is required to find the minimum budget necessary for arc capacity augmentations in G in order to find a feasible flow vector. Formulate this as a network flow problem.

1.64 Let G$= (\mathcal{N}, \mathcal{A})$ be a directed network. **X, Y** are nonempty disjoint subsets of $\mathcal{N}$ in G. **A**$\subset \mathcal{A}$ is said to be an **X, Y-** separating arc set in G, if the deletion of the arcs in **A** leaves no chain from any node in **X** to any node in **Y** in the remaining network. Likewise a subset of nodes **Z** $\subset \mathcal{N} \setminus$ (**X** $\bigcup$ **Y**) is said to be an **X, Y-** separating node set in G, if the deletion of the nodes in **Z** together with all the arcs incident at them leaves no chain from any node in **X** to any node in **Y** in the remaining network. Formulate the problem of finding a minimum cardinality **X, Y-** separating arc and node sets in G as network flow problems. Solve these problems for the network in Figure 1.35 with **X**=$\{1,2,3\}$, **Y**=$\{10,11\}$ using the algorithms discussed in the following chapters.

1.65 There are n modules each of which has to be assigned to one of two available processors. For $j = 1$ to n, c_{j1}, c_{j2} are the costs of

executing module j on processors $1, 2$ respectively. In addition to these processor costs, there is a cost of communication between modules assigned to different processors. So, the communication cost between modules i, j is 0 if both i, j are assigned to the same processor, d_{ij} otherwise. Given c_{j1}, c_{j2}, and d_{ij} for all i, j, formulate the problem of assigning the modules to the two available processors so as to minimize the total cost (processor costs + communication costs) as a minimum capacity cut problem in an undirected network. Construct this model for the problem with $n = 4$, and the other data given below, and obtain an optimum solution.

				d_{ij} for			
		c_{j1}	c_{j2}	$i=1$	2	3	4
$j=$	1	50	20	x	5	20	0
	2	60	30	5	x	34	18
	3	11	15	20	34	x	10
	4	15	10	0	18	10	x

(Dutta, Koehler, and Whinston [1982], Stone [1977])

1.66 Eulerian Trails in Directed Networks Given a directed network, an Eulerian trail in it is a circuit that passes through each arc exactly once. A directed network is said to be Eulerian if there exists an Eulerian trail in it. Prove that a connected directed network is Eulerian iff for every node in the network, its in-degree is equal to its out-degree. Develop a version of the cycle tracing and inserting algorithm of Section $1 \cdot 3 \cdot 8$, to find an Eulerian trail in an Eulerian directed network. (Ebert [1988])

1.67 Dairy Model A co-operative dairy region has n milk processing factories, with factory j having a capacity of b_j KL (Kilolitre)/day of milk input, $j = 1$ to n. It has m milk suppliers with supplier i producing a_i KL milk/day, $i = 1$ to m. Γ_i is the subset of factories to which supplier i can take his milk, $i = 1$ to m. For $i = 1$ to $m, j \in \Gamma_i$, c^1_{ij} is the cost (\$/KL) of transporting milk from supplier i to factory j.

$\triangle_j$ is the subset of other factories that factory j can ship oversupply to, or get supplies from in case of undersupply; $j = 1$ to n, and $c^2_{j_1,j_2}$

is the cost (\$/KL) of shipping between factories j_1, j_2 : $j_1 = 1$ to $n, j_2 \in \triangle_{j_1}$.

The parameter u_j is the number of different process lines at factory j with l_{jt}, k_{jt} as the lower and upper bounds (KL/day) on input and c^3_{jt} as the cost (\$/KL input) of processing at the tth process line $t = 1$ to u_j, $j = 1$ to n. r is the total number of different dairy products produced in the region, and g_{jtw} is the yield of product w in product units/KL milk input into process line t as factory j, $j = 1$ to $n, t = 1$ to $u_j, w = 1$ to r.

It is required to measure the output of each product from each process line in units of KL input milk, according to the average yield from all process lines at all factories put together, rather than in product units, so that all flows can be measured in KL milk units. In terms of these KL milk equivalent units, d^1_w, d^2_w are the lower and upper bounds for daily production and v_w is the return (\$/unit) of product $w, w = 1$ to r.

All milk supply has to be processed on the day of its production. Formulate the problem of determining an optimal allocation plan so as to maximize the net return as a generalized network flow problem. (Mellalieu and Hall [1983])

1.68 School Assignment In a region, there are m school districts with a school in each. For $i = 1$ to m, p_i is the number of excess pupils (if the school there has inadequate capacity, $p_i = 0$ otherwise), and c_i is the excess school capacity (if it does, otherwise $c_i = 0$), in district i. The distance matrix (d_{ij}) between school districts, is given. As far as possible, all pupils will attend the school in their district. All excess pupils from a district with inadequate capacity have to be assigned to a district with excess capacity, but all of them to one school only. (i.e., there should be no splitting of excess pupils from a district between two or more schools).

Formulate the problem of assigning the excess pupils to schools subject to the no split-constraint, so as to minimize the total distance travelled by all the students as a 0-1 integer program. Show that when the integer restrictions or the variables are relaxed, this problem can be solved as a generalized network flow problem. If the "no-splitting" constraint can be ignored, show that the problem becomes a

straightforward transportation problem. Construct both these models for the problem with the following numerical data: $m = 7, p = (p_i) = (23, 18, 30, 20, 0, 0, 0), c = (c_i) = (0, 0, 0, 0, 45, 50, 30)$, and develop an efficient heuristic method based on network models to obtain a reasonable solution to this problem.

$d = (d_{ij}) =$

to j =	5	6	7
from i = 1	14	23	12
2	30	22	10
3	25	14	37
4	20	26	27

(Bovet [1982])

1.69 Operator Scheduling The day is divided into 48 half-hour intervals and a_i is the number of operators required on duty at the telephone company switchboard during the ith half-hour of the day $i = 1$ to 48. Operators work in shifts called tours. Assume that each tour has to be a continuous stretch of 6 or 8 half-hour intervals beginning and ending with any of the half-hours. Let c_{i1}, c_{i2} be the cost of the tour per operator beginning with the ith half-hour, of length 3 or 4 hours respectively, $i = 1$ to 48.

Formulate the problem of determining the number of operators to hire for each possible tour, so as to minimize the cost of meeting the requirements, as a minimum cost flow problem. Construct the model for the following numerical data. Here, the half-hour 1 is 7:00 a.m. to 7:30 a.m.; a_i = 12, 20, 28, 46, 86, 158, 186, 200, 200, 200, 200, 200, 198, 198, 198,192, 186, 184, 178, 158, 128, 114, 92, 102, 124, 118, 117, 116, 104, 104, 104, 104, 108, 50, 50, 16, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, for $i = 1$ to 48; and $c_{i1} = \$17, c_{i2} = \$22$ for tours beginning with 3rd to 23rd half-hours, $c_{i1} = \$20, c_{i2} = \$26$ for tours beginning at 24th to 27th half-hours, $c_{i1} = \$25, c_{i2} = \$33$ for tours beginning at 28th to 48th half-hours.

Assume now that the total duration of a tour may vary from 2 to 9 hours, and always consists of an integer number of half-hours. Tours of duration 8 to 9 hours should include two half-hour breaks. Tours of duration 5 to $7\frac{1}{2}$ hours should include one half-hour break. Tours

of length $4\frac{1}{2}$ hours or less should consist of a continuous stretch. The actual timing of the break periods defines a trick. An ideal trick is one in which the breaks appear between work intervals of equal length. Alternative tricks may be built from the ideal ones by allowing the break periods to be shifted by one half-hour. Associated with each trick is a cost, which is given. Given the demand for operators by half-hours, the problem is to determine the assignment of operators to tricks at minimum cost. Formulate this problem as a linear integer programming problem. Discuss a heuristic approach for solving this problem by solving the network flow model (as in (i) above) for an auxiliary problem where the break periods are ignored temporarily. Then adjust the demands to accommodate break periods in the generated tours, which may create additional demand for operators that needs to be filled again. (Segal [1974])

1.70 Transportation Scheduling In transportation applications, the more difficult problem is to route or schedule vehicles or aircraft to carry out the shipping, a combinatorial optimization problem. In this application there are eight locations, the travel time between location pairs in Airflying Minutes is given within brackets, and number of units to be shipped between them is given in the line below it.

Each aircraft has a capacity of 9,000 units. All shipping has to be completed in one night, starting at 10:00 p.m. and finishing at 5:00 a.m. Each aircraft used may start at any location at 10:00 p.m., and can finish at any location. The stoptime between arrival and departure at each location is a fixed 20 minutes independent of the quantities to be loaded or unloaded or both.

(i) A total of 43,475 units have to be shipped out of location 1. Since the capacity of a plane is 9,000 units, this implies that the number of departures from location 1 has to be $\geqq 5$. Similarly, lower bounds on the total number of departures from each location, and the total number of arrivals at each location, can be obtained. Treating these as the right-hand side constants and the travel times as the costs, formulate the problem of determining a lower bound on the total flying minutes needed in this problem (and

hence a lower bound for the number of planes needed for the task) as a classical transportation problem.

to →	1	2	3	4	5	6	7	8	units leaving
from 1	x	(52)	(71)	(75)	(70)	(115)	(115)	(56)	
		19052	8244	4209	11970	0	0	0	43475
2	(49)	x	(38)	(45)	(56)	(80)	(80)	(85)	
	25729	x	3637	1871	0	0	0	0	31237
3	(68)	(38)	x	(31)	(41)	(43)	(45)	(77)	
	10044	6703	x	5456	0	5021	9264	0	36488
4	(72)	(45)	(31)	x	(70)	(40)	(37)	(90)	
	2641	2234	667	x	0	6807	3836	0	16185
5	(70)	(56)	(41)	(70)	x	(85)	(60)	(36)	
	0	0	0	0	x	0	0	7860	7860
6	(115)	(80)	(43)	(40)	(85)	x	(24)	(120)	
	229	672	3483	2494	0	x	0	0	6878
7	(115)	(80)	(45)	(37)	(60)	(24)	x	(95)	
	1581	0	7908	0	0	0	x	0	9489
8	(56)	(85)	(77)	(90)	(36)	(120)	(95)	x	
	8010	0	0	0	0	0	0	0	8010
units arriving	48234	28661	23939	14030	11970	11828	13100	7860	

(ii) Discuss an approach (based on heuristics or integer programming formulations) of taking the optimum solution of the transportation model in (i); and converting it into actual routes for the planes, to complete the shipping task subject to the constraints stated above, using the smallest number of planes.

(Wolters [1979])

1.71 Vehicle Scheduling

(i) A central office (CO) has to make u round trips. Trip r begins at clock time t_r at the CO and returns back at clock time $T_r = t_r + a_r$,

where a_r is the time duration needed to complete trip r and return to the CO, $r = 1$ to u. The vehicle assigned to make trip r will therefore be available for reassignment to another trip after clock time T_r, if necessary. It is required to find the minimum number of vehicles needed to carry out all these trips. Formulate this as a minimum cost network flow problem.

(ii) Consider a generalization of the above problem in which there are l types of vehicles (small, medium, large, etc.). $\mathbf{V}_r = \{i : \text{type } i$ vehicle is capable of making trip $r\}, r = 1$ to u. $a_{ir}, andc_{ir}$ are the time duration needed for a type i vehicle if it makes the rth trip, and the corresponding cost, $i \in \mathbf{V}_r, r = 1$ to u. It is required to find an assignment of vehicles to trips, so as to minimize the total cost of making all the trips. Discuss how the formulation in (i) can be extended into a "modified" network flow problem, to provide a practical approach for solving this problem.

(Dantzig and Fulkerson [1954], Diez-Canedo and Escalante [1977])

1.72 Matrix with the Consecutive 1's Property Let A be a 0-1 matrix with the property that in each column all the 1's are contiguous. Prove that A is a totally unimodular matrix.

1.73 Let A be an $m \times n$ integer matrix and $b \in \mathbb{R}^n$. Let $\mathbf{P}(b) = \{x : Ax \leqq, b, x \geqq 0\}$. Prove that the following are equivalent:(i) A is totally unimodular,(ii) for each integer vector b and integer $r \geqq 1$, every integer vector in $\mathbf{P}(rb)$ can be expressed on the sum of r integer vectors in $\mathbf{P}(b)$.
(Baum and Trotter [1978])

1.74 Let matrix D have rank r.

(i) If D is a unimodular matrix, prove that every extreme point of

$$Dx = d$$

$$g \geqq x \geqq 0$$

is an integer vector whenever d, g are integer vectors of appropriate dimension (some or all of the components of g could be $+\infty$).

(ii) Let A be a matrix of order $m \times n$, and $\mathbf{Q}(b) = \{x : Ax = b, x \geqq 0\}$. Prove that A is unimodular iff for any integer vector b, integer $h \geqq 1$, and integer vector $\bar{x} \in \mathbf{Q}(hb)$, there exist integer vectors $\bar{x}^t \in \mathbf{Q}(b)$ for $t = 1$ to h such that $\bar{x} = \sum_{t=1}^{h} \bar{x}^t$.

(iii) Let b be an integer vector. If Γ is a subset of $\mathbb{R}^n$, a point $\hat{x} \in \Gamma$ is said to be a *minimal vector in* Γ if there does not exist an $x \in \Gamma$ satisfying $x \leq \hat{x}$. Let $D(b)$ be the matrix whose row vectors constitute the set of all minimal vectors among the set of integral vectors in $\mathbf{Q}(b)$. Consider the following problems.

(1.24)		(1.25)	
maximize	$\pi\ \mathbf{e}$	maximize	$\pi\ \mathbf{e}$
subject to	$\pi D(b) \leq w$	subject to	$\pi D(b) \leq w$
	$\pi \geq 0$		$\pi \geq 0$, and integral

where $\mathbf{e}$ is the vector of all 1's. Prove that for every integer $r \geqq 1$ and integer vector $\bar{x} \in \mathbf{Q}(rb)$, there exist integer vectors $\bar{x}^t \in \mathbf{Q}(b)$ for $t = 1$ to r, such that $\bar{x} = \sum_{t=1}^{r} \bar{x}^t$ iff for every integer vector w; the optimum objective values α, β in (1.24), (1.25) satisfy $\beta = \lfloor \alpha \rfloor$.
(Baum and Trotter [1977])

1.75 $\mathbb{T}$ is a rooted spanning tree in $\mathrm{G} = (\mathcal{N}, \mathcal{A})$. We are given the following: e is an in-tree arc, i, j, are the parent and son nodes on e. $\mathbf{X} = \mathbf{H}(\mathbb{T}, j)$ is the family of node j in $\mathbb{T}$. $\bar{\mathbf{X}} = \mathcal{N} \backslash \mathbf{X}$. Give rigorous proofs of the following.

(i) For each $p \in \mathbf{X}$, the predecessor path of p must pass through node j and actually must include e.

(ii) For each $v \in \bar{\mathbf{X}}$, the predecessor path of v does not pass through node j.

(iii) For any out-of-tree arc in the cut $[\mathbf{X}, \bar{\mathbf{X}}]$, its fundamental cycle with respect to $\mathbb{T}$ must include e.

(iv) For any out-of-tree arc both of whose nodes are either in $\mathbf{X}$ or in $\bar{\mathbf{X}}$, its fundamental cycle with respect to $\mathbb{T}$ does not include e.

1.76 Let A be a 0-1 matrix. It is said to have unique subsequence (or precedence) if its rows can be ordered so that all columns with a one in a particular row have their subsequent, that is next or precedent (i.e., previous) one, if it exists, in a unique common row. Notice that in a matrix with unique subsequence or precedence property, there could be more than two ones in each column. The following is a 0-1 matrix with unique subsequence property with the rows in natural order. Prove that a 0-1 matrix A with unique subsequence or precedence property is totally unimodular. (Ryan and Falkner [1988])

$$\begin{pmatrix} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\ 0&0&1&1&0&1&1&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&0 \\ 0&0&0&1&0&0&1&0&1&0&1&0&1&1&1 \end{pmatrix}$$

1.77 Let G= $(\mathcal{N}, \mathcal{A}, \breve{s}, \breve{t})$ be a connected directed network. Suppose we are given a cut separating $\breve{s}$ and $\breve{t}$ in G as a subset of arcs. Discuss a procedure for identifying a subset of nodes $\mathbf{X} \subset \mathcal{N}$ such that this cut is $[\mathbf{X}, \bar{\mathbf{X}}]$. Is the choice of $\mathbf{X}$ unique? Also, prove that every cut separating $\breve{s}$ and $\breve{t}$ in G is an arc disjoint union of cutsets separating $\breve{s}$ and $\breve{t}$.

1.78 Let G= $(\mathcal{N}, \mathcal{A}, 0, k, V)$ be a connected directed single commodity flow network with $V = (V_1, V_2, \ldots, V_{r+1}, 0, \ldots, 0)^T$ as the vector of exogenous flow values at the nodes, where $V_1 > 0$, and V_2 to V_{r+1} are all < 0. Let $|\mathcal{N}| = n, |\mathcal{A}| = m$; and let $\mathbf{K}$ denote the set of all feasible node-arc flow vectors in G.

Prove that a flow vector $\overline{f} \in \mathbf{K}$ is an extreme point of $\mathbf{K}$ iff the set of arcs $A(\overline{f}) = \{(i,j) : (i,j \in \mathcal{A} \text{ and } 0 < \overline{f}_{ij} < k_{ij}\}$ constitutes a forest in G. If $\overline{f}$ is an extreme point of $\mathbf{K}$, prove that it is a nondegenerate extreme point if the set of arcs $A(\overline{f})$ constitutes a spanning tree in G, degenerate extreme point otherwise.

Let $\overline{f} = (\overline{f}_{ij})$ be an extreme point of $\mathbf{K}$, and $\mathbb{C}$ a simple cycle in G with an orientation. Define $\varepsilon^+(\mathbb{C}, \overline{f}), = \min\{k_{ij} - \overline{f}_{ij} : (i,j)$

is a forward arc on $\mathbb{C}\}, \varepsilon^-(\mathbb{C}, \overline{f}) = \min\{\overline{f}_{ij} : (i,j)$ a reverse arc on $\mathbb{C}\}, \varepsilon(\mathbb{C}, \overline{f}) = \min\{\varepsilon^+(\mathbb{C}, \overline{f}), \varepsilon^-(\mathbb{C}, \overline{f})\}$. Let $\mu(\mathbb{C}) = (\mu_{ij}(\mathbb{C}))$ be the incidence vector of $\mathbb{C}$ given by $\mu_{ij}(\mathbb{C}) = 0$, if (i,j) is not on $\mathbb{C}, +1$, if (i,j) is a forward arc on $\mathbb{C}$, and -1, if (i,j) is a reverse arc on $\mathbb{C}$. If $0 < \varepsilon(\mathbb{C}, \overline{f}) < \infty$, define the new flow vector $f' = \overline{f} + \varepsilon(\mathbb{C}, \overline{f})\mu(\mathbb{C})$. Prove that f' is an adjacent extreme point of $\overline{f}$ on $\mathbf{K}$ iff the following condition 1 holds.

Condition 1: The only cycle in the set of arcs in $A(\overline{f})$ and $\mathbb{C}$ put together, is either $\mathbb{C}$, or $\mathbb{C}$ with its orientation reversed.

Conversely, every adjacent extreme point of $\overline{f}$ on $\mathbf{K}$ is obtained as $\overline{f} + \varepsilon(\mathbb{C}, \overline{f})\mu(\mathbb{C})$ for some simple cycle $\mathbb{C}$ in G satisfying condition 1 and $0 < \varepsilon(\mathbb{C}, \overline{f}) < \infty$.

If $\mathbb{C}$ is a simple cycle satisfying condition 1 and $\varepsilon(\mathbb{C}, \overline{f}) = \infty$, show that $\{\overline{f} + \lambda\mu(\mathbb{C}) : \lambda \geq 0\}$ is an extreme half-line of $\mathbf{K}$ through $\overline{f}$, and conversely every extreme half-line of $\mathbf{K}$ through $\overline{f}$ is obtained in this manner from some simple cycle $\mathbb{C}$ in G satisfying condition 1 and $\varepsilon(\mathbb{C}, \overline{f}) = +\infty$.

If $k = \infty$, prove that $\overline{f} \in \mathbf{K}$ is an extreme point of $\mathbf{K}$ iff $A(\overline{f})$ is a tree with the source node 1 as the root, and the sink nodes 2 to $r+1$ as terminal nodes.

Let $k = \infty$ and $\overline{f}, f'$ be two extreme points of $\mathbf{K}$. Prove that $\overline{f}, f'$ are adjacent iff the arcs in $A(f')\backslash A(\overline{f})$ constitute a path which connects two nodes in $\mathbf{N}$ and does not contain any other nodes of $\mathbf{N}$, where $\mathbf{N}$ is the set of nodes on arcs in $A(\overline{f})$.
(Gallo and Sodini [1979])

1.79 Let f be a feasible flow vector of value $\overline{v}$ in the directed single commodity flow network G= $(\mathcal{N}, \mathcal{A}, 0, k, \check{s}, \check{t}, \overline{v})$ with $k > 0$ and finite. Prove that f is an extreme flow (i.e., a basic feasible flow vector) iff in every simple cycle $\mathbb{C}$ satisfying the property that all the arcs in it carry a positive flow amount in f, there is at least one saturated arc wrt f.

1.80 Prove that a directed network G= $(\mathcal{N}, \mathcal{A})$ is strongly connected iff for every $\emptyset \neq \mathbf{X} \subset \mathcal{N}, \overline{\mathbf{X}} = \mathcal{N}\backslash\mathbf{X}$ with $\overline{\mathbf{X}} \neq \emptyset$, there exists an arc (i,j) in $\mathcal{A}$ with $i \in \mathbf{X}$ and $j \in \overline{\mathbf{X}}$.

1.81 Let G= $(\mathcal{N}, \mathcal{A})$ be a directed connected network. Develop an $O(|\mathcal{A}|)$ algorithm for finding all strongly connected components of G. (Tarjan [1972])

1.82 Consider a finite Markov chain with transition probability matrix P. It is required to identify all the transient states and classify the remaining states into the various closed recurrent classes. Formulate this problem as one of identifying all the strongly connected components in a directed network, and develop an efficient algorithm for it.

Comment 1.2 The first paper using a network or graph model is that of Euler [1736] on the Königsberg bridges problem.

The flow conservation equations (1.18) are also called **Kirchhoff's conservation law of flow** to honor the pioneering work of G. Kirchhoff who formulated them while studying current distribution in electrical networks (in electrical networks, flows satisfying these equations are said to satisfy **Kirchhoff's current (*or* first) law**) in 1847. Although current flows in electrical networks have been studied for a long time, it was not until the 1940's that network flows were used to model and study transportation and distribution problems.

The pioneering book on network flows is that of Ford and Fulkerson [1962], it played a significant role in stimulating research and finding applications for network flow models in many areas. Other books devoted to network flows are Adel'son-Vel'ski, Dinic and Karzanov [1975], Busacker and Saaty [1965], Christofides [1975], Deo [1974], Derigs [1988], Even [1979], Gondran and Minoux [1984], Hu [1969], Iri [1969], Jensen and Barnes [1980], Kennington and Helgasson[1980], Lawler [1976], Minieka [1978], Papadimitriou and Steiglitz [1982], Rockafellar [1984], Swamy and Thulasiraman [1981], and Tarjan [1983]. The book by Bodin, Golden, Assad and Ball [1983] deals with applications in routing. The book by Burkard and Derigs [1980] provides Fortran programs for the special class of matching and assignment problems.

There are many texts in the related areas of graph theory and its applications. Among them we list Berge [1962], Bondy and Murthy [1976], Mayeda [1972], and Wilson [1972]. The book by Lovasz and Plummer [1986] specializes in matching theory; it contains a nice section describing the history of graph theory and network flow theory.

Most of the network problems that we discuss in this book are special cases of linear programming problems, and some of the algorithms discussed are specializations of variants of the simplex method of linear programming. Among the many books on linear programming, we list Bazaraa and Jarvis [1977], Chvatal [1983], Dantzig [1963], Gale [1960], and Murty [1983].

The references listed in this chapter are classified into two parts, books and research publications. The research publications in the second part deal with network models for problems in a variety of areas, algorithms for computing Euler trails, and the chinese postman problem. Several of the exercises given above are taken from these publications.

1.5 References

Books in Network flows and related areas

G. M. ADEL'SON-VEL'SKI, E. A. DINIC, and A. V. KARZANOV, 1975, *Flow Algorithms* (in Russian), Science, Moscow.

A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, 1974, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

A. BACHEM, M. GROTSCHEL and B. KORTE (Eds.), 1982, *Bonn Workshop on Combinatorial Optimization*, North-Holland, Amsterdam.

M. S. BAZARAA and J. J. JARVIS, 1977, *Linear Programming and Network Flows*, Wiley, NY.

C. BERGE, 1962, *The Theory of Graphs*, Methuen, London.

L. BODIN, B. GOLDEN, A. ASSAD, and M. BALL, 1983, *Routing and Scheduling of Vehicles and Crews: State of the Art*, Special issue of *COR*, 10, no. 2, Pergamon Press, NY.

F. BOESCH, 1976, *Large Scale Networks: Theory and Design*, IEEE Press, NY.

J. A. BONDY and U. S. R. MURTHY, 1976, *Graph Theory with Applications*, American Elsevier, NY.

R. E. BURKARD and U. DERIGS, 1980, *Assignment and Matching Problems: Solution Methods with Fortran Programs*, Springer-Verlag, NY.

R. G. BUSACKER and T. L. SAATY, 1965, *Finite Graphs and Networks*, McGraw-Hill, NY.

N. CHRISTOFIDES, 1975, *Graph Theory; an Algorithmic Approach*, Academic Press, NY.

V. CHVATAL, 1983, *Linear Programming*, W. H. Freeman & Co. , NY.

G. B. DANTZIG, 1963, *Linear Programming and Extensions*, Princeton University

Press, Princeton, NJ.
N. DEO, 1974, *Graph Theory with Applications to Engineering and Computer Science*, Prentice- Hall, Englewood Cliffs, NJ.
U. DERIGS, 1988, *Programming in Networks and Graphs*, Lecture notes in Economics and Mathematical Systems 300, Springer -Verlag, NY.
S. E. ELMAGHRABY, 1970, *Some Network Models in Management Science*, Springer-Verlag, NY.
S. EVEN, 1979, *Graph Algorithms*, Computer Science press, Potomac, MD.
L. R. FORD and D. R. FULKERSON, 1962, *Flows in Networks*, Princeton University Press, Princeton, NJ.
D. GALE, 1960, *The Theory of Linear Economic Models*, McGraw-Hill, NY.
G. GALLO and C. SANDI (Eds.), 1986, *Netflow at Pisa, MPS*, 26.
M. R. GAREY and D. S. JOHNSON, 1980, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., NY, 2nd printing.
M. GONDRAN and M. MINOUX, 1984, *Graphs and Algorithms*, Wiley-Interscience, NY.
E. HOROWITZ and S. J. SAHNI, 1978, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD.
T. C. HU, 1969, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA.
M. IRI, 1969, *Network Flow, Transportation and Scheduling*, Academic Press, NY.
P. A. JENSEN and J. W. BARNES, 1980, *Network Flow Programming*, Wiley, NY.
J. KENNINGTON and R. HELGASON, 1980, *Algorithms for Network Programming*, Wiley, NY.
D. KLINGMAN and J. M. MULVEY, (Eds.), 1981, *Network Models and Associated Applications, MPS*,15.
E. L. LAWLER, 1976, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, NY.
L. LOVASZ and M. D. PLUMMER, 1980, *Matching Theory*, North-Holland, Amsterdam.
W. MAYEDA, 1972, *Graph Theory*, Wiley-Interscience, NY.
E. MINIEKA, 1978, *Optimization Algorithms for Networks and Graphs*, Marcel Dekker, NY.
K. G. MURTY, 1976, *Linear and Combinatorial Programming*, Krieger, Malabar, FL.
K. G. MURTY, 1983, *Linear Programming*, Wiley, NY.
C. H. PAPADIMITRIOU and K. STEIGLITZ, 1982, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.
R. T. ROCKAFELLAR, 1984, *Network Flows and Monotropic Optimization*, Wiley-Interscience, NY.
M. N. S. SWAMY and K. THULASIRAMAN, 1981, *Graphs, Networks, and Algorithms*, Wiley-Interscience, NY.
R. E. TARJAN, 1983, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Math. SIAM, 44.

R. J. WILSON, 1972, *Introduction to Graph Theory*, Oliver and Boyd, Edinburgh.

Other References

B. M. BAKER, Sept. 1984, "A Network Flow Algorithm for Project Selection," *JORS*, 35, no. 9 (847-852).

K. R. BAKER, 1976, " Work Force Allocation in Cyclical Scheduling Problems: A Survey," *ORQ*, 27, no. 1,ii (155-167).

M. L. BALINSKI, 1970, "On a Selection Problem," *MS*, 17(230-231).

S. BAUM and L. E. TROTTER, Jr., 1978, "Integer Rounding and Polyhedral Decomposition for Totally Unimodular Systems," (15-23) in R. Henn, B. Korte, and W. Oettli (Eds.), *Arbeitstagung- über Operations Research und Optimierung*, Springer-Verlag, Berlin.

A. A. BERTOSSI, P. CARRARESI, and G. GALLO, 1987, "On Some Matching Problems Arising in Vehicle Scheduling Models," *Networks*, 17, no. 3(271-281).

J. BOVET, Aug. 1982, "Simple Heuristics for the School Assignment Problem," *JORS*, 33, no. 8(695-703).

R. CHANDRASEKARAN and S. SUBBA RAO, May-June 1977, "A Special Case of The Transportation Problem," *OR*, 25, no. 3(525-528).

M. CHESHIRE, K. I. M. McKINNON, and H. P. WILLIAMS, Aug. 1984, " The Efficient Allocation of Private Contractors to Public Works," *JORS*, 35, no. 8(705-709).

G. B. DANTZIG and D. R. FULKERSON, 1954, "Minimizing the Number of Tankers to Meet a Fixed Schedule," *NRLQ*, 1(217-222).

M. D. DIVINE and W. G. LESSO, April 1972, "Models for the Minimum Cost Development of Oil Fields," *MS*, 18, no. 8(B-378-387).

J. M. DIEZ-CANEDO and O. M-M. ESCALANTE, 1977, "A Network Solution to a General Vehicle Scheduling Problem," *EJOR*, 1(255-261).

R. C. DORSEY, T. J. HODGSON, and H. D. RATLIFF, 1975, "A Network Approach to a Multi-facility Multi-product Production Scheduling Problem Without Back Ordering," *MS*, 21(813-822).

A. DUTTA, G. KOEHLER, and A. WHINSTON, Aug.1982, "On Optimal Allocation in a Distributed Processing Environment," *MS*, 28, no. 8(839-853).

J. EBERT, June 1988, "Computing Eulerian Trails," *IPL*, 28, no. 2(93-97).

J. EDMONDS and E. JOHNSON, 1973, "Matching, Euler Tours and the Chinese Postman's Problem," *MP*, 5(88-124).

L. EULER, 1736, "Solutio Problematis ad Geometriam Situs Pertinentis," *Com-*

mun. Acad. Sci. Imp. Petropol., 8(128-140): Opera Omnia(1), Vol. 7.

A. M. FARLEY, July 1980, "Levelling Terrain Trees: A Transshipment Problem," *IPL*, 10, nos. 4/5(189-192).

C. O. FONG, Oct. 1980, "Planning for Industrial Estate Development in a Developing Economy," *MS*, 26, no. 10(1061-1067).

A. GALLO and C. SODINI, 1979, "Adjacent Extreme Flows and Application to Minimum Concave Cost Flow Problems," *Networks*, 9(95-121).

B. GAVISH and P. SCHWEITZER, 1974, "An Algorithm for Combining Truck Trips," *TS*, 8(13-23).

B. GAVISH, P. SCHWEITZER and E. SHLIFER, 1978, "Assigning Buses to Schedules in a Metropolitan Area," *COR*, 5(129-138).

P. R. HALMOS and H. E. VAUGHAN,1950, "The Marriage Problem," *American Journal of Mathematics*, 72(214-215).

I. HELLER and C. B. TOMPKINS, 1958, "Integral Boundary Points of Convex Polyhedra," (247-254) in H. W. Kuhn and A. W. Tucker (Eds.), *Linear Inequalities and Related Systems*, Princeton University Press, Princeton, NJ.

A. J. HOFFMAN and J. B. KRUSKAL, 1958, "Integral Boundary Points of Convex Polyhedra," (223-246)in H. W. Kuhn and A. W. Tucker (Eds.), *Linear Inequalities and Related Systems*, Princeton University Press, Princeton, NJ.

M. IRI, S. AMARI, and M. TAKATA, 1968, "Algebraical and Topological Theory and Methods in Linear Programming with Weak Graphical Representation," (421-464) in K. Kondo (Ed.), *RAAG Memoirs of the Unifying Study of Basic Problems in Engineering and Physical Sciences by Means of Geometry*, 4, G-iX, Gakujutsu Bunken Fukyukai, Tokyo.

W. JACOBS, 1954, "The Caterer Problem," *NRLQ*, 1(154-165).

R. R. LOVE, Jr., and R. R. VEMUGANTI, Jan.-Feb. 1978, "The Single Plant Mold Allocation Problem with Capacity and Changeover Restrictions," *OR*, 26, no. 1(159-165).

T. L. MAGNANTI and R. T. WONG, 1984, "Network Design and Transportation Planning: Models and Algorithms," *TS*, 18(1-56).

U. MANBER and S. ISRANI, 1984, "Pierce Point Minimization and Optimal Torch Path Determination in Flame Cutting," *Journal of Manufacturing Systems*, 3, no. 1(81-89).

P. G. MARLIN, 1981, "Application of the Transportation Model to a Large Scale Districting Problem," *COR*, 8, no. 2(83-96).

P. J. MELLALIEU and K. R. HALL, June 1983, "An Interactive Planning Model

for the New Zealand Dairy Industry," *JORS*, 34, no. 6(521-532).
E. MINIEKA, July 1979, "The Chinese Postman Problem for Mixed Networks," *MS*, 25 no. 7(643-648).
J. PINTO PAIX$\tilde{A}$O and I. M. BRANCO, 1987, "A Quasi-Assignment Algorithm for Bus Scheduling," *Networks*, 17, no. 3(249-269).
M. QUEYRANNE, 1982, "The Tankering Problem," CBA Working Paper Series, University of Houston, Houston.
D. M. RYAN and J. C. FALKNER, June 1988, "On the Integer Properties of Scheduling Set Partitioning Models," *EJOR*, 35, no. 3(442-456).
J. RHYS, Nov. 1970, "A Selection Problem of Shared Fixed Costs and Network Flows," *MS*, 17, no. 3(200-207).
M. SEGAL, July-Aug. 1974, "The Operator Scheduling Problem: A Network Flow Approach," *OR*, 22, no. 4(808-823).
V. SRINIVASAN, Jan.1979, "Network Models for Estimating Brand-Specific Effects in Multi-Attribute Marketing Models," *MS*, 25, no. 1(11-21).
H. S. STONE, Jan. 1977, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," *IEEE Transactions on Software Engineering*, SE-3(85-93).
W. SZWARC and M. E. POSNER, Nov.-Dec. 1985, "The Caterer Problem," *OR*, 33, no. 6(1215-1224).
F. B. TALBOT and J. H. PATTERSON, July 1978, " An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource Constrained Scheduling Problems," *MS*, 24, no. 11(1163-1174).
R. E. TARJAN, 1972, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal of Computing*, 1(146-160).
M. TSO, 1986, "Network Flow Models in Image Processing," *JORS*, 37, no. 1(31-34).
A. F. VEINOTT, Jr., and H. M. WAGNER, 1962, "Optimal Capacity Scheduling-I and II," *OR*, 10, no. 4(518-546).
H. P. WILLIAMS, 1982, "Models with Network Duals," *JORS*, 33(161-169).
J. A. M. WOLTERS, 1979, "Minimizing the Number of Aircraft for a Transportation Network," *EJOR*, 3(394-402).

Index

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