Optimization Algorithms for Networks and Graphs

Second Edition
Revised and Expand

James R. Evans
Edward Minieka

Optimization Algorithms for Networks and Graphs

Second Edition, Revised and Expanded

James R. Evans
University of Cincinnati
Cincinnati, Ohio

Edward Minieka
The University of Illinois at Chicago
Chicago, Illinois

Marcel Dekker New York

Library of Congress Cataloging-in-Publication Data

Evans James R. (James Robert).
Optimization algorithms for networks and graphs. – 2nd ed., rev. and expanded / James R. Evans, Edward Minieka.
p. cm.
Rev. ed. of: Optimization algorithms for networks and graphs / Edward Minieka. c1978.
Includes bibliographical references and index.
ISBN 0-8247-8602-5
1. Graph theory. 2. Network analysis (Planning) 3. Algorithms. I. Evans, James R. II. Minieka, Edward. Optimization algorithms for networks and graphs. III. Title.
QA166.E92 1992
658.4'032–dc20 92-3323
CIP

Marcel Dekker, Inc. and the author make no warranty with regard to *NETSOLVE for the IBM PC and Compatibles*, its accuracy, or its suitability for any purpose other than specified in the manual. This software is licensed solely on an "as is" basis. The only warranty made with respect to *NETSOLVE for the IBM PC and Compatibles* disk is that the disk medium on which the software is recorded is free of defects. Marcel Dekker, Inc. will replace a disk found to be defective if such defect is not attributable to misuse by the purchaser or the purchaser's agent. The defective disk must be returned within ten (10) days to:

Customer Service
Marcel Dekker, Inc.
Post Office Box 5005
Cimarron Road
Monticello, NY 12701
(914) 796-1919

Comments, suggestions, or bug reports concerning *NETSOLVE for the IBM PC and Compatibles* are welcome and should be sent to:

Upstate Resources, Inc.
P.O. Box 152
Six Mile, SC 29682

IBM PC is a registered trademark of International Business Machines Corporation.

COPYRIGHT © 1992 by MARCEL DEKKER, INC. ALL RIGHTS RESERVED
270 Madison Avenue, New York, New York 10016

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage and retrieval system, without permission in writing from the publisher.

NETSOLVE for the IBM PC and Compatibles:
Copyright © 1985–1992 by James P. Jarvis and Douglas R. Shier. All rights reserved.
The manual and disk contained herein may not be reproduced in whole or in part without written permission of James P. Jarvis or Douglas R. Shier.

PREFACE TO THE SECOND EDITION

The 1970s ushered in an exciting era of research and applications of networks and graphs in operations research, industrial engineering, and related disciplines. Network optimization has been an important area of research and application in its own right and, in addition, is increasingly important as a component of broader and more powerful decision support systems.

This is a text about optimization algorithms for problems that can be formulated on graphs and networks. The first edition of this text was unique in providing a comprehensive, cohesive, and clear treatment of this body of knowledge. Many new results and applications have appeared in the literature since that time. This edition provides many new applications and algorithms while maintaining the classic foundations on which contemporary algorithms have been developed.

The major changes in this edition are described below.

1. The original chapters have been expanded and updated and include new material that originated over the past decade and a half. The latter includes such topics as partitioning algorithms for shortest paths, preflow-push algorithms for maximum flow, extensions of the postman problem, the network simplex method, and heuristic procedures for the traveling salesman problem. In addition, the introductory section of each chapter now includes many new and realistic application examples.

2. A new chapter on computer representation, algorithms, heuristics, and computational complexity has been added. The chapter on flow algorithms has been divided into two separate chapters: one on minimum-cost flow algorithms, and one dealing exclusively with maximum-flow algorithms.

3. A computer software package, NETSOLVE, developed by James Jarvis and Douglas Shier, has been integrated with the text to provide students with the ability to solve larger applications than one can expect to by hand.

Every effort has been made to retain the style and flavor of the first edition. The audience consists of advanced undergraduate or beginning graduate students in a variety of disciplines; an operations research or mathematics background is, of course, helpful but not essential. Many sections that rely on prior knowledge of linear programming can be skipped easily. The focus is on an intuitive approach to the inner workings, interdependencies, and applications of the algorithms. Their place in the hierarchy of advanced mathematics and the details of their computer coding are not stressed. The text can be treated comprehensively in a one-semester course or, less thoroughly but without significant omissions, in a one-quarter course.

We gratefully acknowledge the helpful comments provided by James P. Jarvis and Douglas R. Shier, as well as their permission to use NETSOLVE and examples from *Practical Aspects of Network Analysis*, copyright ©1986, 1989 by the authors.

James R. Evans
Edward Minieka

PREFACE TO THE FIRST EDITION

This is not another graph theory text; it is a text about algorithms—optimization algorithms for problems that can be formulated in a network or graph setting.

As a thesis student studying these algorithms, I became aware of their variety, elegance, and interconnections and also of the acute need to collect and integrate them between two covers from their obscure hiding places in scattered journals and monographs. I hope that I have been able in the confines of this text to make a stab at a comprehensive, cohesive, and clear treatment of this body of knowledge.

This text is self-contained at the level of an advanced undergraduate or beginning graduate student in any discipline. An operations research or mathematics background is, of course, helpful but hardly essential.

The text aims at an intuitive approach to the inner workings, interdependencies, and applications of the algorithms. Their place in the hierarchy of advanced mathematics and the details of their computer coding are not stressed.

Chapter 1 contains background information and definitions. Aside from Chapter 1, I have tried to make all chapters as independent of one another as possible, except where one algorithm uses another as a subroutine. Even in this situation, the reader can continue if he is willing to accept the subroutine algorithm on faith. This text can be treated comprehensively in a one-semester course and less thoroughly, but without significant omissions, in a one-quarter course.

It takes a lot of ink to go from source to sink. I wish to thank Randy Brown at Kent State University, Ellis Johnson at IBM, George Nemhauser at Cornell University, and Douglas Shier at the National Bureau of Standards for their careful readings and suggestions. Also, emphatic thanks to my colleague Leonard Kent for his confidence in this project and for years of encouragement. Lastly, thanks to my students who graciously endured three years of classroom testing of the various manuscript stages. Most of all, this book is for you and for your successors, everywhere.

Edward Minieka

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

CONTENTS

1
INTRODUCTION TO GRAPHS AND NETWORKS

1.1 INTRODUCTION AND EXAMPLES

Graph theory is a branch of mathematics that has wide practical application. Numerous problems arising in such diverse fields as psychology, chemistry, industrial and electrical engineering, transportation planning, management, marketing, and education can be posed as problems from graph theory. Because of this, graph theory is not only an area of interest in its own right but also a unifying basis from which results from other fields can be collected, shared, extended, and disseminated.

Unlike other scientific fields, graph theory has a definite birthday. The first paper on graphs was written by the Swiss mathematician Leonhard Euler (1707–1783) and was published in 1736 by the Academy of Science in St. Petersburg. Euler's study of graphs was motivated by the so-called Königsberg bridge problem. The city of Königsberg (now called Kaliningrad) in East Prussia was built at the junction of two rivers and the two islands formed by them (see Fig. 1.1). In all, there were seven bridges connecting the islands to each other and to the rest of the city. Euler was posed the following question: Could a Königsberger start from home and cross each bridge exactly once and return home? The answer, which Euler proved, is no. We shall see why later (Chapter 8) when we study a generalized version of this problem called the postman problem.

The growth of graph theory continued in the late nineteenth and early twentieth centuries with advances motivated by molecular theory and electrical theory. By the 1950s, the field had taken two essentially different directions: the *algebraic* aspects of graph theory and the *optimization* aspects of graph theory. The latter were greatly advanced by the advent of the computer and the discovery of linear programming techniques. This text is concerned almost exclusively with the optimization aspects of graph theory.

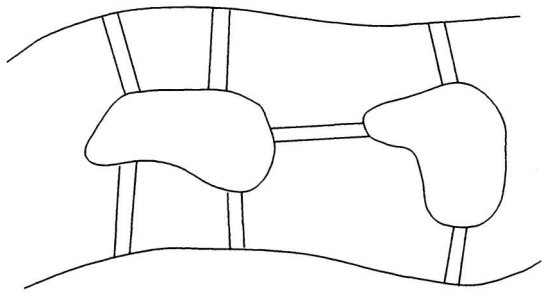


Fig. 1.1 The Königsberg bridge problem.

What is a graph? A graph consists of two parts, points and lines joining these points. The points can be depicted as points in a plane or, if you prefer, points without any specific physical location. For example, in the Königsberg bridge problem of Fig. 1.1, let us label each of the regions of land by the points 1, 2, 3, and 4 and join a line between two points whenever a bridge connects them. The graph corresponding to the Königsberg bridge problem is shown in Fig. 1.2. In keeping with standard terminology, we shall refer

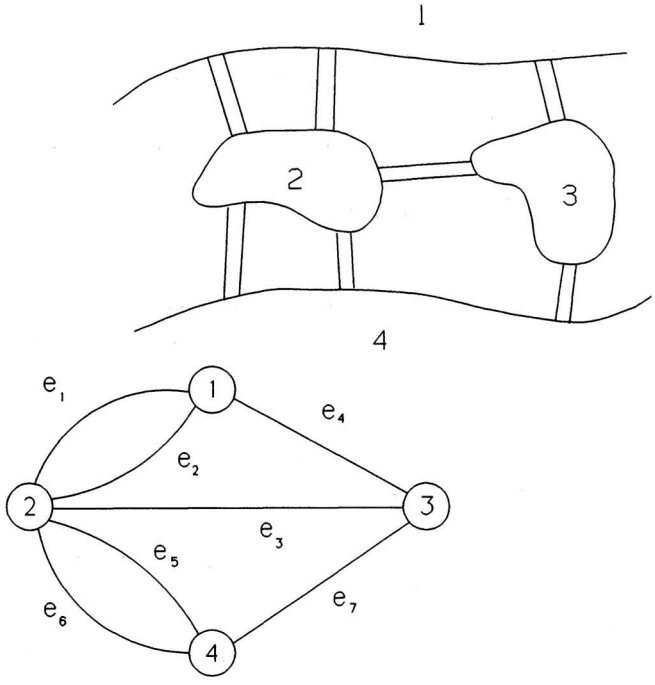


Fig. 1.2 Graph of the Königsberg bridge problem.

to the points of a graph as *vertices* and the lines of a graph as *edges*. Each edge often is given an identifier such as e_1, e_2, . . . or a letter of the alphabet (a, b, . . .).

The same graph could be specified without using a picture simply by listing the vertices 1, 2, 3, 4 and listing the edges $e_1 = (1, 2)$, $e_2 = (1, 2)$, $e_3 = (2, 3)$, $e_4 = (1, 3)$, $e_5 = (2, 4)$, $e_6 = (2, 4)$, $e_7 = (3, 4)$ as pairs of points. From any such listing, we can always draw a picture of the graph. For instance, suppose we provide a list of vertices 1, 2, 3, 4, 5 and edges $a = (1, 3)$, $b = (1, 4)$, $c = (2,3)$, $d = (3, 5)$, $e = (5, 1)$, $f = (5, 3)$. The graph corresponding to these sets is given in Fig. 1.3.

With this motivation, we can now state a formal definition for a graph:

> A *graph* G is a set X whose elements are called *vertices* and a set E of pairs of vertices, called *edges*. The graph G is denoted by (X, E).

Thus, in the previous example, $X = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 3), (1, 4), (2, 3), (3, 5), (5, 1), (5, 3)\}$. Throughout, we shall assume that both the set X and the set E contain only a finite number of elements.

Any situation in which you can list a set of elements and a relationship between *pairs* of elements can be depicted as a graph. For example, let X denote the set of street intersections in a city. Let E denote the set of all street segments that join the intersections. Clearly, (X, E) is a graph. Edges need not correspond to physical objects such as bridges or highways, nor must their lengths be proportional to actual physical distances. As another example, let X denote the set of all airports in Illinois. Let E denote the set of all pairs of airports (x, y) such that there is a nonstop commercial flight between airport x and airport y. Then (X, E) is a graph. Throughout this book we shall see many more examples of problems that can be modeled as graphs.

Whenever the set E consists of *unordered* pairs of vertices, we have an *undirected graph*. In an undirected graph, an edge (x, y) and an edge (y, x) are indistinguishable; we are only concerned with the endpoints of each edge. Both the graphs in Figs. 1.2 and 1.3 are undirected.

Some graphs have more than one edge between a given pair of vertices. In Fig. 1.3, for example, we see that there are two edges between vertices 3 and 5. Since these edges

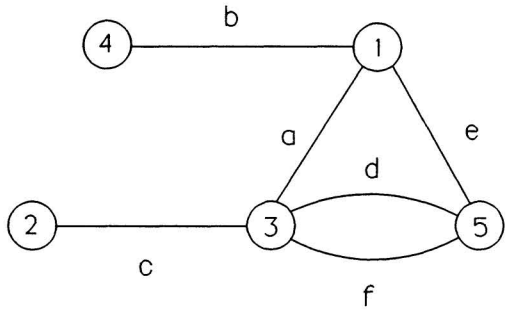


Fig. 1.3 An example of a graph.

are undirected, either of them can be written as (5, 3) or as (3, 5). Whenever we have multiple edges between the same two vertices, we can denote them by a subscripted pair of vertices if we wish to distinguish between them. Thus, in Fig. 1.3, we could denote edge d by $(3, 5)_1$ and edge f by $(3, 5)_2$. When no confusion will develop, we shall omit the subscripts.

In many practical situations, however, such as when we have one-way streets, the *direction* of an edge must be specified. We specify direction in a graph by drawing arrows instead of lines between the vertices. Directed edges are often called *arcs*, and the graph is called a *directed graph*. Vertices in directed graphs are more commonly called *nodes*. The set of arcs is denoted by A. An example of a directed graph is shown in Fig. 1.4. The set $A = \{(1, 2), (2, 1), (2, 3), (2, 4), (4, 3), (3, 1)\}$. Notice that in this case the arcs (1, 2) and (2, 1) are not identical, but represent two different ordered pairs of vertices. If (x, y) is a directed arc, then x is called the *tail* of the arc and y is called the *head* of the arc. As an example of a directed graph, let X denote the set of all passengers aboard a certain transatlantic flight. Let A denote the set of all pairs (x, y) of passengers such that passenger x is older than passenger y and both speak a common language. Clearly, (X, A) is a directed graph with node set X and arc set A. Is it possible for this graph to have both an arc (x, y) and an arc (y, x)?

In this book, we shall normally use the terms *vertex* and *edge* when referring to an undirected graph and the terms *node* and *arc* when referring to a directed graph. We will use the notation (X, E) to denote an undirected graph with vertex set X and edge set E, and we will use the notation (X, A) to denote a directed graph with node set X and arc set A. Unless otherwise stated, we will normally use m to denote the number of vertices or nodes, and n to denote the number of edges or arcs. Occasionally, however, an algorithm or concept might apply to either a directed or an undirected graph. In these cases, either pair of terms can be used, but no confusion should result in context. We caution you that many of the definitions used throughout this book are not uniform in the literature on graphs and networks. Thus, you must be careful when reading other references on the subject.

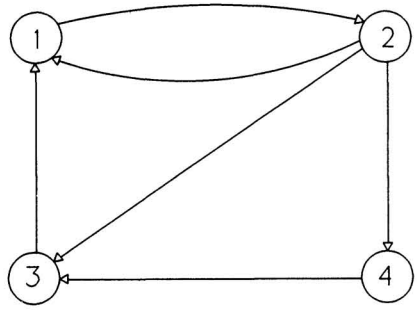


Fig. 1.4 An example of a directed graph.

A *network* is a graph with one or more numbers associated with each edge or arc. These numbers might represent distances, costs, reliabilities, or other relevant parameters.

Graph and network optimization represents one of the most important areas of operations research for several reasons. Graphs and networks can be used to model many diverse applications such as transportation systems, communication systems, vehicle routing, production planning, and cash flow analysis. They are more readily accepted by nontechnical people since they are often related to the physical system, which enhances model understanding. Finally, graphs and networks have special properties that allow the solution of much larger problems than can be handled by any other optimization technique and are much more computationally efficient than other types of solution procedures.

Each remaining chapter of this text is devoted to a single type of practical optimization problem on a graph or network. The procedures that are given for seeking optimum solutions to these problems are called *algorithms*; hence, the title of this text. Algorithms in general will be discussed in more detail in Chapter 2. As some of these optimization algorithms build upon others, the order of presentation is restricted, and these considerations have dictated the sequencing of the chapters of this text.

1.2 SOME ESSENTIAL CONCEPTS AND DEFINITIONS

To decrease the dependence between chapters, some basic concepts and definitions that are needed throughout are presented here. The motivation for these definitions will be reserved to later chapters where applications are discussed in greater depth. Unless specified otherwise, these definitions apply to both directed and undirected graphs.

We assume that the reader is familiar with basic set notation such as set membership ($\in$), union ($\cup$), intersection ($\cap$), and subset ($\subseteq$). If S is a set, then $S \cup \{e\}$ will be written as $S + e$. If $e \in S$, then $S - \{e\}$ will be written as $S - e$. $|S|$ denotes the *cardinality* of the set S, that is, the number of elements in S. Finally, $S \oplus T = S \cup T - S \cap T$ is the *symmetric difference* of sets S and T.

An edge that has both of its endpoints as the same vertex is called a *loop*. In Fig. 1.5, edge e_2 is a loop. (Most graphs that we shall consider in this text will contain neither loops nor multiple edges. This will be assumed unless otherwise stated.) If a graph is loopless, does not contain multiple edges, and $|X| = m$ and $|E| = n$ (that is, the graph has m

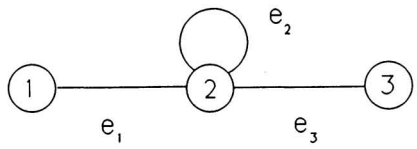


Fig. 1.5 A graph containing a loop.

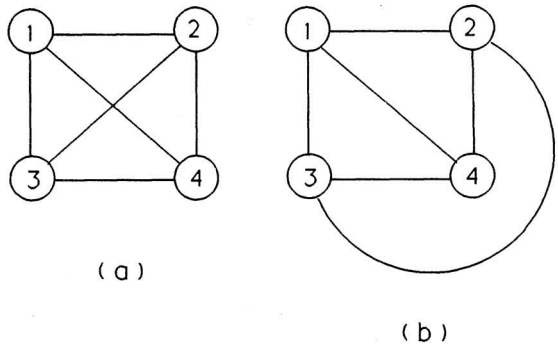


Fig. 1.6 Example of a planar graph.

vertices and n edges), then $n \leq m\ (m - 1)/2$. Why? A graph in which every pair of vertices are connected by an edge is called a *complete graph.*

A graph is *planar* if it *can* be drawn with no two edges crossing each other. The graph in Fig. 1.6a is planar even though it is drawn with two edges crossing each other. Figure 1.6b shows how it can be redrawn in a different fashion.

Every planar graph G has a *geometric dual*, which we denote by G^*. The dual graph is constructed by placing a vertex in every region of the original graph and connecting the new vertices by drawing an edge across every original edge. Figure 1.7 shows the construction of a dual graph. The dual of a dual graph always results in the original graph.

A vertex and an edge are said to be *incident to one another* if the vertex is an endpoint of the edge. Thus in Fig. 1.8 edge e_1 is incident with vertex 1 and with vertex 2. Two edges are said to be *adjacent* if they are both incident to the same vertex. In Fig. 1.8, edges e_1 and e_2 are adjacent because they are both incident with vertex 2. Two vertices are said to be *adjacent* to one another if there is an edge joining them. In Fig. 1.8, vertices 2 and 3 are adjacent to one another because there is an edge (e_2) that joins them.

The *degree* of a vertex is the number of edges incident with it. Thus, in Fig. 1.8 the degree of vertex 2 is three. For directed graphs, we could also define the *in-degree* of a node as the number of arcs directed toward it and the *out-degree* of a node as the number of arcs directed from it. Thus, in Fig. 1.9 the degree of node 2 is 3, the in-degree is 2, and the out-degree is 1.

Consider any sequence $x_1, x_2, \ldots, x_n, x_{n+1}$ of vertices. A *path* is any sequence of edges $e_1, e_2, \ldots, e_n$ such that the endpoints of edge e_i are x_i and x_{i+1} for $i = 1, 2, \ldots, n$. Vertex x_1 is called the *initial vertex* of the path; vertex x_{n+1} is called the *terminal vertex* of the path. The path is said to extend from its initial vertex to its terminal vertex. The *length* of the path equals the number of edges in the path. In Fig. 1.8, the sequence of edges e_1, e_2, e_3 forms a path of length 3 from vertex 1 to vertex 4.

The concept of a path in a directed graph is the same as in an undirected graph. That is, it does not matter what the directions of the arcs are in a path in a directed graph; either

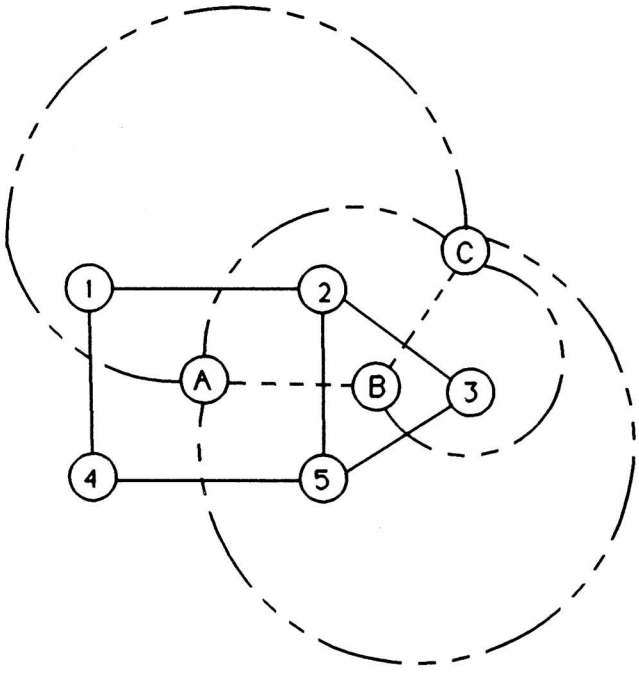


Fig. 1.7 Construction of the geometric dual graph. (——) G; (---) G^*.

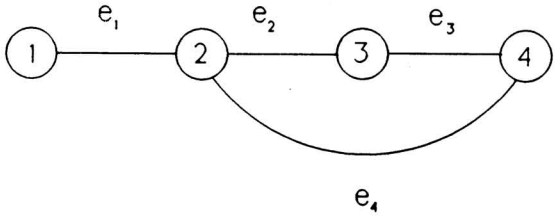


Fig. 1.8 An undirected graph.

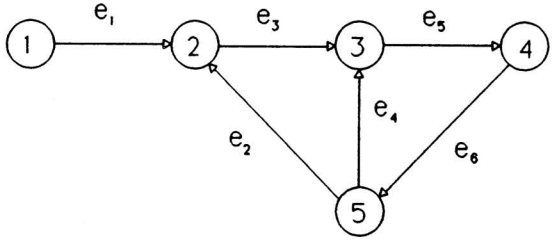


Fig. 1.9 A directed graph.

$e_i = (x_i, x_{i+1})$ or $e_i = (x_{i+1}, x_i)$. However, a *directed path* is a path for which $e_i = (x_i, x_{i+1})$ for $i = 1, 2, \ldots, n$. In a directed path, all arcs are pointed in the same direction from the initial vertex to the terminal vertex. In Fig. 1.9, a path would be the sequence of arcs e_1, e_2, e_4, e_5, whereas arcs e_1, e_3, e_5 would form a directed path.

A *cycle* is a path whose initial vertex and terminal vertex are identical. In Fig. 1.8, edges e_2, e_3, e_4 would be an example of a cycle. A *directed cycle* is a directed path whose initial vertex and terminal vertex are identical. Thus, for a directed graph, a directed cycle has all arcs pointed in the same direction, whereas in a cycle the arcs can be oriented in either direction. The length of a cycle is defined as the length of its corresponding path. Figure 1.9 has a cycle consisting of the arcs e_2, e_3, and e_4 and a directed cycle consisting of the arcs e_4, e_5, and e_6. Can you find another directed cycle? A directed graph is called *acyclic* if it contains no directed cycles.

A path or cycle (whether directed or not) is called *simple* if no vertex is incident to more than two of its edges (that is, if the path or cycle properly contains no cycles). In Fig. 1.9, the path e_1, e_2 is simple whereas the path e_1, e_3, e_4, e_2 is not.

A graph is *connected* if there is a path joining every pair of distinct vertices in the graph. For example, the graphs of Figs. 1.8 and 1.9 are connected, but the graph in Fig. 1.10 is not connected since there is no path joining any vertex in the set $\{1, 3, 4\}$ to any in the set $\{2, 5, 6\}$. A graph may be regarded as consisting of a set of connected graphs. Each of these connected graphs is called a *component* of the original graph. Thus the graph in Fig. 1.10 has two components.

A graph g is a *subgraph* of a graph G if all the vertices and edges of g are in G and each edge of g has the same two end vertices that it has in G.

Let X' be any subset of X, the vertex set of the graph (X, E). The graph whose vertex set is X' and whose edge set consists of all edges in E with both endpoints in X' is called the *subgraph generated by* X'. In a similar fashion, let E' be any subset of E, the edge set of the graph (X, E). The graph whose edge set is E' and whose vertex set consists of all vertices incident to an edge in E' is called the *subgraph generated by* E'. For example, the subgraph generated by the vertices $\{1, 2, 3\}$ in Fig. 1.8 is shown in Fig. 1.11. The subgraph generated by the arcs $\{e_1, e_2, e_3\}$ in Fig. 1.9 is shown in Fig. 1.12.

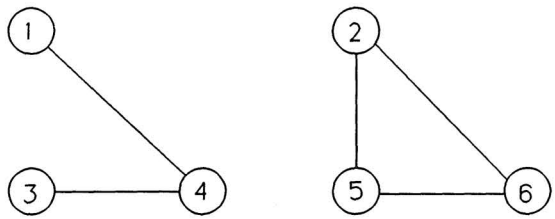


Fig. 1.10 A disconnected graph.

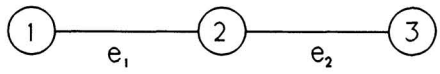


Fig. 1.11 Subgraph generated by a set of vertices in Fig. 1.8.

A graph is called a *tree* if it satisfies two conditions:

1. The graph is connected.
2. The graph contains no cycles.

In Fig. 1.8, the subgraphs generated by the following sets of edges each form a tree: $\{e_1, e_3, e_4\}$, $\{e_3, e_4\}$, $\{e_1, e_2, e_4\}$, $\{e_3\}$. The edges $\{e_1, e_2, e_3, e_4\}$ and their incident vertices do not form a tree since the subgraph generated contains a cycle.

A *forest* is any graph that contains no cycles. Thus, a forest consists of one or more trees. A *spanning tree* of a graph is any tree that includes every vertex in the graph. In Fig. 1.9 the subgraph generated by arcs $\{e_1, e_2, e_3, e_5\}$ forms a spanning tree since it includes each vertex 1, 2, 3, 4, and 5. Clearly, no spanning tree can exist for a graph with more than one component, and every connected graph possesses a spanning tree. An edge of a spanning tree is called a *branch*; an edge in the graph that is not in the spanning tree is called a *chord*.

A tree with one edge contains two vertices; a tree with two edges contains three vertices; a tree with three edges contains four vertices; and, in general, a tree with $m - 1$ edges must contain m vertices. Hence, each spanning tree of a (connected) graph with m vertices consists of $m - 1$ edges.

Since a tree is necessarily connected, there is at least one path between any two vertices. Suppose there were more than one path between a pair of vertices. Taking these together, we would create a cycle. But a tree has no cycles. Therefore, there must be a *unique path* between any two vertices in a tree.

A set of edges in a connected graph whose removal from the graph increases the number of components in the graph is called a *disconnecting set*. If the addition of any edge of a disconnecting set necessarily decreases the number of components in the remaining graph, the disconnecting set is called a *cutset*. A cutset that contains as a proper

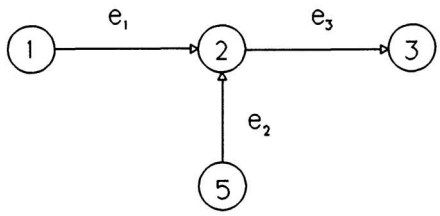


Fig. 1.12 Subgraph generated by a set of arcs in Fig. 1.9.

subset no other cutset is called a *simple cutset* (also called a *proper cutset* or *minimal disconnecting set*). In Fig. 1.3, the set of edges $\{a, b, c\}$ form a disconnecting set since the number of components in the remaining graph is increased to three. However, this is not a cutset since if edge a is readded to the graph, the number of components remains three. The set $\{b, c\}$ is a cutset of the graph, but is not a simple cutset since $\{b\}$ and $\{c\}$ alone are also cutsets.

A *flow* is a way of sending objects from one place to another in a network. For example, the shipment of finished goods from a manufacturer to a distributor, the movement of people from their homes to places of employment, and the delivery of letters from their point of posting to their destinations can all be regarded as flows. Many important problems involve flows in networks. For example, one might wish to maximize the amount transported from one place to another, or might wish to determine the least-cost way to send a given number of objects from their sources of supply to their destinations.

The objects that travel or flow through a network are called *flow units* or *units*. Flow units can be finished goods, people, letters, information, or almost anything. Nodes from which units begin their journey through a network are called *source nodes*. Nodes to which units are routed are called *sink nodes*. Source nodes usually have a *supply*, which represents the number of units available at the node. Sink nodes usually have a *demand*, representing the number of units which must be routed to them.

In a flow problem, each arc often has a *capacity* associated with it. The capacity of an arc is the maximum allowable flow that may travel over it. Arc capacities are simply upper bounds on the flow units.

The fundamental equation governing flows in networks is known as *conservation of flow*. Simply stated, conservation of flow states that at every node

$$\text{Flow out} - \text{flow in} = 0$$

The supply at a source node is considered as "flow in"; the demand at a sink node is considered as "flow out." To illustrate this, consider the network in Fig. 1.13. Node 1 is a source node with a supply of 10; nodes 2 and 4 are sink nodes with demands of 5 each. Node 3 has neither supply nor demand; it is called a *transshipment node*. Let $f(x,y)$ denote the flow from node x to node y. The conservation of flow equations are written as

$$\text{Node 1:} \quad f(1, 2) + f(1, 3) - 10 = 0$$

$$\text{Node 2:} \quad 5 + f(2, 4) - f(1, 2) - f(4, 2) = 0$$

$$\text{Node 3:} \quad f(3, 4) - f(1, 3) = 0$$

$$\text{Node 4:} \quad 5 + f(4, 2) - f(2, 4) - f(3, 4) = 0$$

If we rewrite these equations, putting all constants on the right-hand side, we obtain

$$f(1, 2) + f(1, 3) = 10$$
$$f(2, 4) - f(1, 2) - f(4, 2) = -5$$
$$f(3, 4) - f(1, 3) = 0$$
$$f(4, 2) - f(2, 4) - f(3, 4) = -5$$

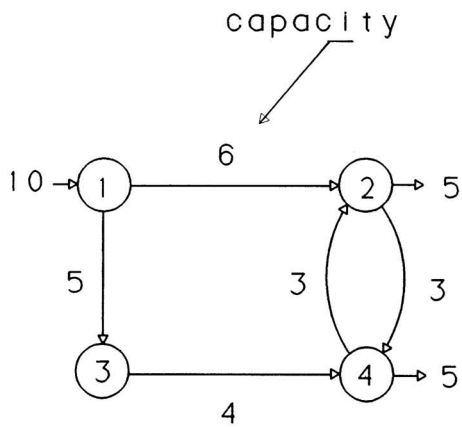


Fig. 1.13 Illustration of conservation of flow.

Notice that the supplies will be positive numbers whereas demands will be negative numbers. Also, note that each variable appears in exactly two equations, one with a $+1$ coefficient and one with a -1 coefficient. The node (equation) corresponding to the $+1$ corresponds to the tail of the arc, while the node corresponding to the -1 corresponds to the head of the arc.

The arc capacities in Fig. 1.13 can be written as the constraints

$$f(1, 2) \leq 6$$
$$f(1, 3) \leq 5$$
$$f(3, 4) \leq 4$$
$$f(2, 4) \leq 3$$
$$f(4, 2) \leq 3$$

A flow is called *feasible* if it satisfies the conservation of flow and capacity constraints. Since the conservation of flow equations and capacity constraints are linear, you might suspect that flow problems involve linear programming. Indeed, they are one of the most important classes of linear programming problems, as we shall see in a later chapter.

1.3 MODELING WITH GRAPHS AND NETWORKS

As we have noted, graphs and networks can be used to model many important problems in engineering, business, and the physical and social sciences. In this section we shall present several examples of how graphs and networks can be useful modeling tools.

A Scheduling Problem

A university is sponsoring a half-day seminar on network applications. There are six speakers, each planning to speak one hour. If they were scheduled in six different time slots, the seminar would extend beyond the planned four-hour period. Because of nonoverlapping interests, certain speakers can be scheduled simultaneously. The following matrix shows which speakers *should not* be scheduled at the same time.

	1	2	3	4	5	6
1					x	
2						
3				x	x	x
4			x		x	x
5	x		x	x		x
6			x	x	x	

We can model this problem as a graph by letting the speakers correspond to vertices. An edge is drawn between two vertices if the speakers should not be scheduled at the same time. This is shown in Fig. 1.14. A solution exists if the vertices can be labeled A, B, C, and D (corresponding to the four time slots) such that no two adjacent vertices have the same label. A related problem would be to find the minimum number of time slots necessary to schedule all speakers.

Offshore Pipeline Design

An oil company owns several oil drilling platforms in the Gulf of Mexico. Oil that is recovered from each platform must be shipped to refineries in Louisiana. A network of pipelines can be constructed between the platforms and the Louisiana shore. How should the pipeline network be designed to minimize the construction cost?

Each drilling platform can correspond to a vertex of a graph. The edges in the graph can be assigned weights representing the distance between platforms. One vertex also corresponds to the refinery. The problem then becomes one of finding a connected subgraph having the minimum total distance. The graph in Fig. 1.15 illustrates a small example.

Clearly, the pipeline network must be connected. We claim that the optimal solution must be a spanning tree. If it were not, then it would contain a cycle. This means that there would be two different ways to transport the oil from one vertex to another on the cycle. Clearly, one could remove some edge from the cycle and reduce the total cost while still retaining a connected graph. Can you find the minimum-cost spanning tree in Fig. 1.15?

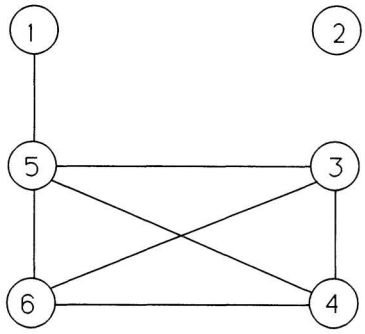


Fig. 1.14 Graph for the speaker scheduling problem.

Delivering "Meals on Wheels"

"Meals on Wheels" is a service which delivers prepared lunches to people who are unable to shop or cook for themselves. The meals are prepared early in the morning and delivered from a services center to the individual locations between 10 a.m. and 2 p.m. The program director needs to decide how many drivers are needed and in what order the meals should be delivered to ensure that all will be delivered within the specified time window.

The problem can be modeled as a graph by letting each delivery location as well as the services center correspond to a vertex on a graph. Edges connecting the vertices represent the shortest route that can be driven between the locations. Each edge can be

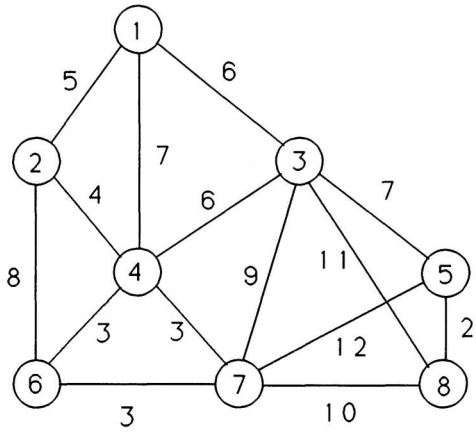


Fig. 1.15 An oil pipeline network.

assigned the travel time between the incident vertices. The problem is to determine a set of cycles that pass through the services center vertex such that the sum of the edge weights in each cycle does not exceed four hours. To minimize the cost of fuel, one might seek the set of feasible cycles having the smallest sum of the edge weights. Another reasonable problem would be to determine the minimum number of drivers needed so that all cycles are feasible.

Facility Layout

Graph theory has useful applications in facility layout. We can always represent any layout as a planar graph as shown in Fig. 1.16. Notice that this graph is very similar to the concept of a dual graph. We associate a vertex with each department in the layout and draw an edge between two vertices if the departments are adjacent. The more difficult problem is to find an appropriate graph in order to design a layout.

Suppose that we wish to lay out m facilities, for example, different departments in a library. A traffic study might be conducted, resulting in the construction of a matrix of specifying the number of trips made between each pair of facilities. This matrix is as follows:

		1	2	3	4	5
1.	Entrance	—				
2.	Catalogue	200	—			
3.	Photocopy	4	77	—		
4.	Journals	80	125	64	—	
5.	New books	32	42	19	26	—

We would like to locate the departments so as to maximize the sum of the values of *adjacent* pairs of departments.

We may represent each department by a vertex of a complete graph. The number of trips between each pair of departments will be assigned as a "weight" of the corresponding edge of the graph (Fig. 1.17). The solution is to find the planar subgraph that contains all the vertices and has maximum total edge weight. Figure 1.18 shows an example of a feasible solution. Can you find a better solution? Of course, the graph theoretic solution must be scaled to reflect the appropriate size of the departments.

Equipment Replacement

The operating and maintenance costs of a piece of equipment increase with age. Thus we want to replace the equipment periodically before the costs become excessive. We assume that we know the costs over the life of the equipment and can determine the salvage value at the end of each year. Let c_{ij} represent the total cost of purchasing a new piece of

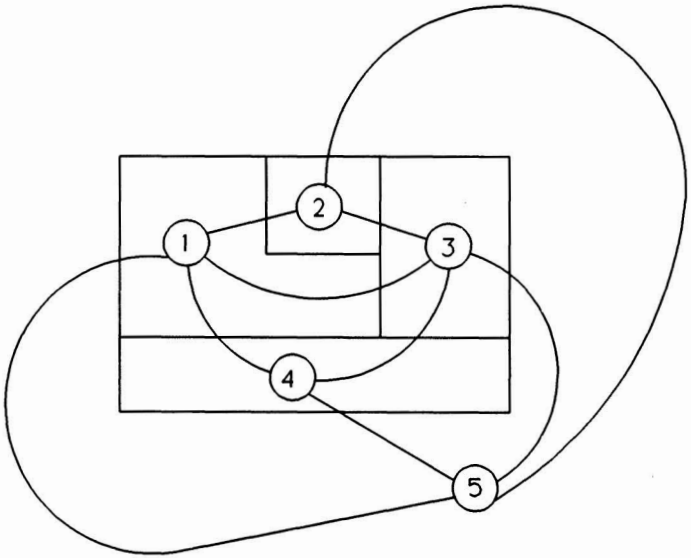


Fig. 1.16 A department layout and its planar graph.

equipment at the beginning of period i and selling it at the beginning of period j. This value would consist of the purchase cost, plus all operating and maintenance costs for periods i through $j - 1$, less the salvage value at the end of period $j - 1$.

We can find the optimal replacement plan over a time horizon of n periods by letting each period correspond to a node of a network. An arc from node i to node j represents purchasing the equipment at the beginning of period i and keeping it through period $j - 1$.

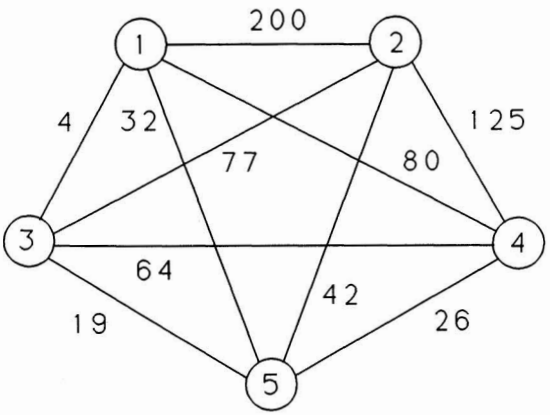


Fig. 1.17 Complete graph for facility layout problem.

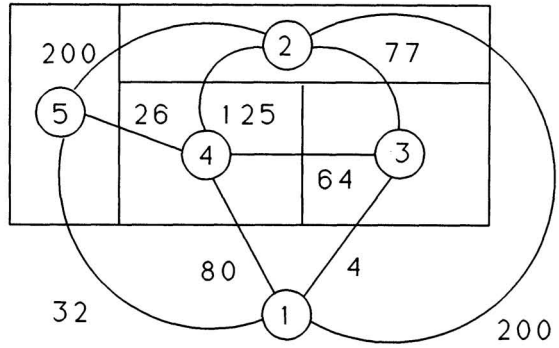


Fig. 1.18 Feasible solution to facility layout problem.

Associated with each arc is the total cost for this policy. The network for a three-year period is shown in Fig. 1.19. Any path from node 1 to node 4 represents a sequence of purchasing and selling decisions. For example, the path (1, 2), (2, 4) represents purchasing a new machine at the beginning of year 1, holding it for one year, replacing it at the beginning of year 2, and holding this machine for two years. Since each arc has a cost associated with it, the shortest path through the network will provide the minimal-cost policy.

Employment Scheduling

In many industries, the demand for services varies considerably over time. One strategy for reducing costs is to employ part-time workers for short periods of time. The minimum number of employees required in each time period is known. We seek to balance the cost of hiring and firing with the expense of having some idle employees on the payroll for a short period of time. Suppose that the requirements for three periods are 15, 10, and 20. Let x_{ij} be the number of employees hired at the beginning of period i and retained through period $j - 1$ and c_{ij} be the cost per employee associated with this strategy. Thus x_{12} represents hiring for period 1 only; x_{14} represents hiring for periods 1, 2, and 3 together. Finally, let s_j be the excess number of workers employed in period j.

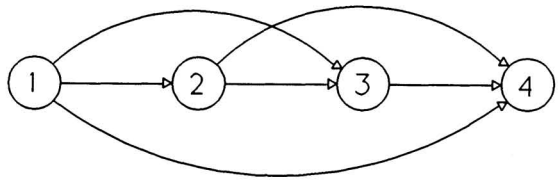


Fig. 1.19 Equipment replacement problem network.

The number of workers available in period 1 is

$$x_{12} + x_{13} + x_{14}$$

Using the relationship that the number available less the excess must equal the requirements, we have

$$x_{12} + x_{13} + x_{14} - s_1 = 15$$

Similarly, for periods 2 and 3 we have

$$x_{13} + x_{14} + x_{23} + x_{24} - s_2 = 10$$

and

$$x_{14} + x_{24} + x_{34} - s_3 = 20$$

If we subtract the first equation from the second, and the second equation from the third, and multiply the third equation by -1, we have the following

$$\begin{array}{rrrrrrrrrrrrrrr}
x_{12} & + & x_{13} & + & x_{14} & & & & & - & s_1 & & & = & 15 \\
-x_{12} & & & & & + & x_{23} & + & x_{24} & + & s_1 & - & s_2 & = & -5 \\
 & - & x_{13} & & & - & x_{23} & & & + & s_2 & - & s_3 & = & 10 \\
 & & & - & x_{14} & & & - & x_{24} - x_{34} & & & + & s_3 & = & -20
\end{array}$$

Each column on the left-hand side has exactly one $+1$ and one -1; this is characteristic of all network flow problems. Through algebraic manipulation we have created a set of equations that represents the node-arc incidence matrix of a network flow problem. If we associate a node with each equation, we can draw a directed arc corresponding to each column from the row with the $+1$ to the row with the -1. This leads to the network in Fig. 1.20. The sign of the right-hand side values determines whether the nodes are sources or sinks. If we assign a cost to each arc, we can find the flow that minimizes the total cost.

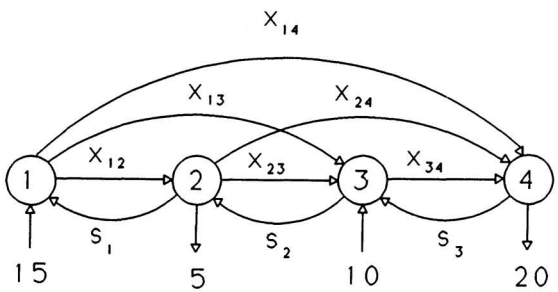


Fig. 1.20 Employment scheduling network.

These examples show only a fraction of the types of problems that can be modeled and solved using graphs and networks. Throughout this book you will see many more examples.

APPENDIX: LINEAR PROGRAMMING

Many of the network optimization problems considered in subsequent chapters can be formulated as linear programming problems. Often, insights from the linear programming formulation can assist in developing efficient solution procedures. This section gives a brief review of linear programming results that will be needed later in this book. The presentation here, however, is by no means intended to produce a profound or intuitive understanding of linear programming. A more complete treatment of linear programming can be found in the references.

Basic Concepts and Theory

The general form of a linear program is

$$\text{Minimize} \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{1}$$

subject to

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{2}$$

$$x_1, x_2, \cdots, x_n \geq 0 \tag{3}$$

In this linear program, $x_1, x_2, \ldots, x_n$ represent the unknown decision variables; $c_1, c_2, \ldots, c_n$ are cost coefficients; $b_1, b_2, \ldots, b_m$ are the right-hand side values; and the a_{ij}, $i = 1$ to m and $j = 1$ to n, are called the technological coefficients. Expression (1) is called the objective function, (2) is called the constraint set, and (3) are nonnegativity constraints. Any solution that satisfies all constraints is called a *feasible solution.*

In this formulation, the constraint set is written in equality form. In general, linear programming constraints may have $\geq$ or $\leq$ relations but can always be converted to equalities by the addition of slack variables. Also, the objective function (1) can be expressed as a maximization instead of minimization.

Two other mathematical representations of the formulation given above are often used for notational convenience. One way is to express the problem in summation notation:

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \qquad i = 1, 2, \ldots m$$
$$x_j \geq 0, \qquad j = 1, 2, \ldots n$$

A second way is to use matrix notation. Let $\mathbf{c} = [c_1, c_2, \ldots, c_n]$, $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, $\mathbf{b} = [b_1, b_2, \ldots, b_m]$, $\mathbf{0}$ be an n-dimensional vector of zeros, and A be the matrix of the technological coefficients a_{ij}. An equivalent formulation is (we assume that vectors are transposed appropriately for the proper multiplication)

$$\text{Minimize} \quad \mathbf{cx}$$
subject to
$$A\mathbf{x} = \mathbf{b}$$
$$\mathbf{x} \geq \mathbf{0}$$

Every linear program (called the *primal problem*) has a related problem called the *dual problem*. We associate a dual variable u_i with each primal constraint. If the primal objective is to minimize, the dual objective is maximization. The sign restrictions on the dual variables are determined by the type of primal constraint. If the primal problem is minimization, then an equality constraint results in u_i being unrestricted; if the primal constraint is $\geq$, the $u_i \geq 0$; if the primal constraint is $\leq$, then $u_i \leq 0$. The dual to problem (1)–(3) is

$$\text{Maximize} \quad b_1 u_1 + b_2 u_2 + \cdots + b_m u_m \tag{1'}$$

subject to

$$a_{11} u_1 + a_{21} u_2 + \cdots + a_{m1} u_m \leq c_1 \tag{2'}$$
$$a_{12} u_1 + a_{22} u_2 + \cdots + a_{m2} u_m \leq c_2$$
$$\vdots$$
$$a_{1n} u_1 + a_{2n} u_2 + \cdots + a_{mn} u_m \leq c_n$$
$$u_1, u_2, \cdots, u_m \text{ unrestricted in sign} \tag{3'}$$

In terms of the alternate notation, this dual problem can be expressed in summation notation:

$$\text{Maximize} \quad \sum_{i=1}^{m} b_i u_i$$

subject to

$$\sum_{i=1}^{m} a_{ij} \leq c_j, \qquad j = 1, \ldots n$$

u_i unrestricted

or in matrix notation:

Maximize $\mathbf{bu}$

subject to

$$\mathbf{u}A \leq \mathbf{c}$$

$\mathbf{u}$ unrestricted

Suppose we know a feasible solution for a linear program and a feasible solution for its dual problem. A fundamental result in linear programming is that these primal and dual feasible solutions are both optimal if they satisfy *complementary slackness conditions*. Using the matrix notation, we have

$$\mathbf{u}(A\mathbf{x} - \mathbf{b}) = \mathbf{u}A\mathbf{x} - \mathbf{ub} = 0$$

and

$$(\mathbf{c} - \mathbf{u}A)\mathbf{x} = \mathbf{cx} - \mathbf{u}A\mathbf{x} \geq 0$$

It follows that $\mathbf{ub} = \mathbf{u}A\mathbf{x} \leq \mathbf{cx}$; that is, the value of the dual objective function (max) is always less than or equal to the value of the primal objective function (min). If $\mathbf{ub} = \mathbf{cx}$ then clearly $\mathbf{x}$ and $\mathbf{u}$ must be optimal to their respective problems. This is true whenever $(\mathbf{c} - \mathbf{u}A)\mathbf{x} = 0$. The complementary slackness conditions are

$$\mathbf{u}(A\mathbf{x} - \mathbf{b}) = 0 \quad \text{and} \quad (\mathbf{c} - \mathbf{u}A)\mathbf{x} = 0$$

Any feasible solutions satisfying these conditions will be optimal.

The Simplex Algorithm

The simplex algorithm is a method for solving linear programs by systematically moving from one solution to another until the optimal solution is found (or the problem is determined to be unbounded or infeasible). Consider the problem

$$\begin{aligned} \min \quad z &= \mathbf{cx} \\ A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

We can partition A into a set of m linearly independent columns called the basis matrix B and the remaining columns called the nonbasis matrix N. Letting $\mathbf{x}_B$ and $\mathbf{x}_N$ be the associated variables, the problem can be written as

$$\begin{aligned} \min \quad z &= \mathbf{c}_B\mathbf{x}_B + \mathbf{c}_N\mathbf{x}_N \\ B\mathbf{x}_B + N\mathbf{x}_N &= \mathbf{b} \\ \mathbf{x}_B, \mathbf{x}_N &\geq 0 \end{aligned}$$

Solving the constraint set for $\mathbf{x}_B$ and substituting, we obtain

$$\begin{aligned} \min \quad z &= \mathbf{c}_B B^{-1}\mathbf{b} + (\mathbf{c}_N - \mathbf{c}_B B^{-1} N)\mathbf{x}_N \\ \mathbf{x}_B + B^{-1}N\mathbf{x}_N &= B^{-1}\mathbf{b} \\ \mathbf{x}_B, \mathbf{x}_N &\geq \mathbf{0} \end{aligned}$$

If $\mathbf{x}_N = \mathbf{0}$, the solution is called a *basic feasible solution*. By defining $\mathbf{c}_B B^{-1}\mathbf{b}$ as z_0, $B^{-1}\mathbf{b}$ as $\mathbf{b}'$, $\mathbf{c}_B B^{-1}N$ as the vector $\mathbf{z}$, the columns of $B^{-1}N$ as the vectors $\mathbf{y}_j$, and R as the index set of the variables $\mathbf{x}_N$, we have

$$\begin{aligned} \min \quad z &= z_0 + \sum_{j \in R} (c_j - z_j)x_j \\ \mathbf{x}_B + \sum_{j \in R} \mathbf{y}_j x_j &= \mathbf{b}' \\ \mathbf{x}_B, \mathbf{x}_N &\geq \mathbf{0} \end{aligned}$$

The solution $\mathbf{x}_B = \mathbf{b}'$ is optimal if $c_j - z_j \geq 0$ for all $j \in R$. If not, then choose the index k with the most negative value of $c_j - z_j$. The variable x_k will enter the basis and replace one of the current basic variables. We determine which variable to leave the basis by finding the minimum $\{b_i'/y_{ik},\ y_{ik} > 0\}$, where b_i' is the ith component of $\mathbf{b}'$. This is the maximum value that x_k can be increased until a basic variable is reduced to zero. Suppose this occurs for the index r. Then the rth basic variable will leave the basis. We *pivot* to a new basic feasible solution by setting $x_k = b_r'/y_{rk}$ and $x_{Bi} = b_i' - (b_r'/y_{rk})y_{ik}$ for i not equal to r. The column corresponding to x_k replaces the column corresponding to x_{Br} in the basis B and the procedure is repeated.

To summarize, the steps of the simplex algorithm are

1. *Priceout*: Determine which nonbasic variable should enter the basis to improve the objective function.

2. *Ratio test*: Determine which current basic variable would leave the basis first as the value of the new nonbasic variable is increased.
3. *Pivot*: Exchange the entering and exiting variables in the basis and adjust the values of all remaining basic variables.

In Chapter 6 we shall see how properties of networks enable the simplex algorithm to be streamlined very efficiently.

EXERCISES

1. Consider the graph shown in Fig. 1.21. Formally list the sets X and E that define this graph.

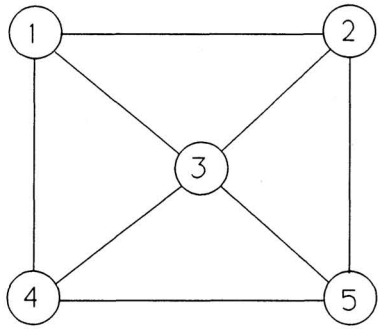


Fig. 1.21 Graph for Exercise 1.

2. Draw the graph corresponding to the following sets: $X = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (1, 3), (1, 6), (2, 4), (2, 6), (5, 6)\}$.
3. Is a five-vertex complete graph planar? What can you say about complete graphs with more than five vertices?
4. Construct the dual graph for the graph in Exercise 1 (Fig. 1.21).
5. Show that for any graph G the number of vertices with odd degree is even.
6. Show that for any directed graph, the sum of the out-degrees must equal the sum of the in-degrees.
7. Find all simple cycles in the graph of Exercise 1 (Fig. 1.21).
8. For the graph in Fig. 1.21, construct the subgraph generated by the vertices $\{1, 2, 3, 4\}$.

9. For the graph in Fig. 1.21, construct the subgraph generated by the edges (1, 4), (3, 4), (3, 5), and (4, 5).
10. For the graph in Fig. 1.21, find five different spanning trees.
11. Is it possible for a cutset and a cycle to contain exactly one edge in common? Why?
12. Is there a relationship between cutsets in a graph and cycles in its dual graph? If so, what is it?
13. Construct a graph whose vertex set is the set of courses you are required to pass for your degree. Place an arc from vertex x to vertex y if course x is a prerequisite for course y. Give an interpretation for each of the following:
    (a) Path
    (b) Directed path
    (c) Cycle
    (d) Directed cycle
    (e) Connected component
14. Is every subset of a tree also a tree? Is every subset of a forest also a forest?
15. Show that if T is a spanning tree, then $T + e$, where e is a chord, contains exactly one cycle.
16. Suppose forest F consists of t trees and contains v vertices. How many edges are in forest F?
17. Let the subgraph generated by edges $\{e_1, e_2, e_3, e_4\}$ in Fig. 1.22 define a spanning tree. The set of cycles formed by adding each chord to the tree, one at a time, is called a *fundamental set of cycles*. Show that any cycle in the graph can be expressed as the symmetric difference of some subset of fundamental cycles. Does this have a familiar analogy with a well-known result in linear algebra?

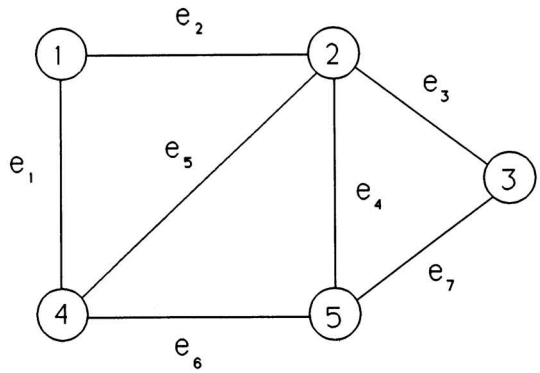


Fig. 1.22 Graph for Exercise 17.

18. Any partition of vertices into two sets generates a cutset. For a graph of n vertices, how many such cutsets can be found? Illustrate your answer with the graph in Fig. 1.21. Does this enumerate all possible cutsets?

19. Many of the original 13 states share common borders. Construct a graph G whose vertices represent the original 13 states. Join two vertices by an edge if the corresponding states have a common border. Is this graph connected? Find the cutset with the smallest number of edges. Is there any state which, if it did not join the Union, would have severed all land travel between the remaining states?

20. Write down the conservation of flow and capacity constraints for the network in Fig. 1.23.

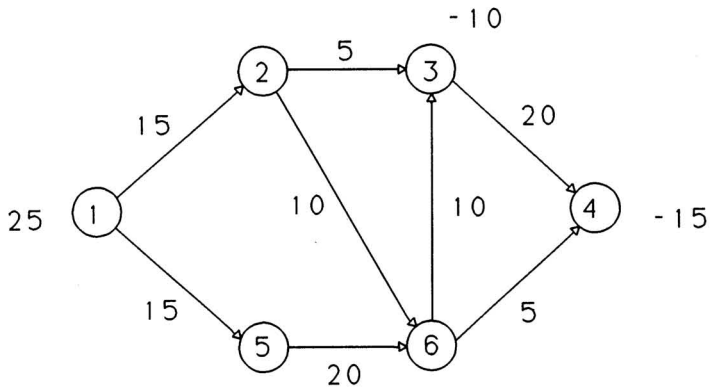


Fig. 1.23 Network for Exercise 20.

21. Is the graph in Fig. 1.23 acyclic? Why or why not?

22. Are the conservation of flow equations for a network linearly independent? (Assume that the sum of the supplies equals the sum of the demands.) Why or why not?

23. Consider the following linear programming problem:

Maximize

$$3x_1 + 2x_2 + 1x_3 - 4x_4$$

such that

$$2x_1 + 1x_2 + 3x_3 + 7x_4 \leq 10$$

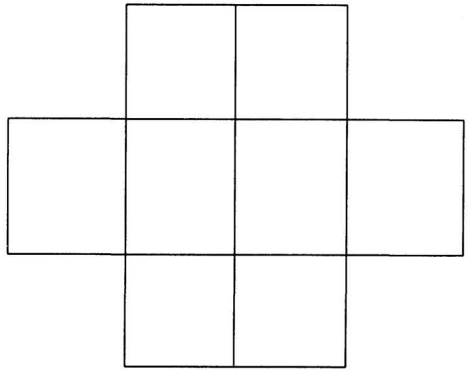
$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0$$

(a) Find an optimal solution. (Hint: Consider only basic solutions.)
(b) What is the dual linear programming problem?

(c) Use the complementary slackness conditions to find an optimal solution to the dual.

24. A classic problem is the "knight's tour." On an $n \times n$ chessboard, starting from some square, move the knight so that it lands on every space exactly once. Show how a graph can be used to solve this problem. Illustrate on a 4×4 chessboard.

25. Show how to use a graph to solve the following puzzle. Place the numbers 1 through 8 in the boxes so that no two consecutive numbers are horizontally, vertically, or diagonally adjacent to each other.



26. A caterer must provide n_j napkins for d successive days. He can buy new napkins at c dollars each or launder the dirty napkins. The slow laundry takes 2 days and costs a dollars per napkin; the fast laundry takes 1 day and costs b dollars per napkin. Model this problem as a network flow problem to meet demands at minimum total cost.

27. A product can be produced in each of T time periods. The product can be either sold or held until the next time period. The demand for period t is D_t. All demands must be met, the unit cost of production in period t is p_t, and the unit cost of carrying inventory from one period to the next is h_t. Show how to construct a network flow model to determine the optimal production and inventory policy.

REFERENCES

Bartholdi, J. J., L. K. Platzman, R. L. Collins, and W. H. Warden, III, 1983. A Minimal Technology Routing System for Meals on Wheels, *Interfaces*, *13*(3), pp. 1–8.

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

Bennington, G. E., 1974. Applying Network Analysis, *Ind. Eng. 6*(1), pp. 17–25.
Berge, C., 1973. *Graphs and Hypergraphs* (translated by E. Minieka), North-Holland, Amsterdam.
Busacker, R. and T. Saaty, 1965. *Finite Graphs and Networks*, McGraw-Hill, New York.
Chachra, V., P. M. Ghare, and J. M. Moore, 1979. *Applications of Graph Theory Algorithms*, North-Holland, New York.
Dantzig, G. B., 1963. *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.
Deo, N., 1974. *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Englewood Cliffs, New Jersey.
Ford, L. R., and D. R. Fulkerson, 1962. *Flows in Networks*, Princeton University Press, Princeton, New Jersey.
Foulds, L. R., and H. V. Tran, 1986. Library Layout Via Graph Theory, *Comput. Ind. Eng. 10*(3), pp. 245–252.
Frank, H., and I. Frisch, 1971. *Communication, Transmission, and Transportation Networks*, Addison-Wesley, Reading, Massachusetts.
Hadley, G., 1962. *Linear Programming*, Addison-Wesley, Reading, Massachusetts.
Harary, F., 1969. *Graph Theory*, Addison-Wesley, Reading, Massachusetts.
Hu, T. C., 1969. *Integer Programming and Network Flows*, Addison-Wesley, Reading, Massachusetts.
Ore, O., 1963. *Graphs and Their Uses*, Random House New Mathematical Library, Random House, New York.
Potts, R. B., and R. M. Oliver, 1972. *Flows in Transportation Networks*, Academic Press, New York.
Simonnard, M., 1966. *Linear Programming* (translated by W. S. Jewell), Prentice-Hall, Englewood Cliffs, New Jersey.
Swamy, M. N. S., and K. Thulasiraman, 1981. *Graphs, Networks, and Algorithms*, Wiley, New York.
Wilson, R., 1972. *Introduction to Graph Theory*, Academic Press, New York.
Wu, N., and R. Coppins, 1981. *Linear Programming and Extensions*, McGraw-Hill, New York.

Introduction to Graphs and Networks

Bartholdi, J. J. , L. K. Platzman , R. L. Collins , and W. H. Warden , 1983. A Minimal Technology Routing System for Meals on Wheels, Interfaces, 13(3), pp. 18.
Bazaraa, M. S. , J. J. Jarvis , and H. D. Sherali 1990. Linear Programming and Network Flows, 2nd ed., Wiley, New York.
26 Bennington, G. E. , 1974. Applying Network Analysis, Ind. Eng. 6(1), pp. 1725.
Berge, C. 1973. Graphs and Hypergraphs (translated by E. Minieka), North-Holland, Amsterdam.
Busacker, R. and T. Saaty 1965. Finite Graphs and Networks, McGraw-Hill, New York.
Chachra, V. , P. M. Ghare , and J. M. Moore 1979. Applications of Graph Theory Algorithms, North-Holland, New York.
Dantzig, G. B. 1963. Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey.
Deo, N. 1974. Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, Englewood Cliffs, New Jersey.
Ford, L. R. , and D. R. Fulkerson 1962. Flows in Networks, Princeton University Press, Princeton, New Jersey.
Foulds, L. R. , and H. V. Tran , 1986. Library Layout Via Graph Theory, Comput. Ind. Eng. 10(3), pp. 245252.
Frank, H. , and I. Frisch 1971. Communication, Transmission, and Transportation Networks, Addison-Wesley, Reading, Massachusetts.
Hadley, G. 1962. Linear Programming, Addison-Wesley, Reading, Massachusetts.
Harary, F. 1969. Graph Theory, Addison-Wesley, Reading, Massachusetts.
Hu, T. C. 1969. Integer Programming and Network Flows, Addison-Wesley, Reading, Massachusetts.
Ore, O. 1963. Graphs and Their Uses, Random House New Mathematical Library, Random House, New York.
Potts, R. B. , and R. M. Oliver 1972. Flows in Transportation Networks, Academic Press, New York.
Simonnard, M. 1966. Linear Programming (translated by W. S. Jewell), Prentice-Hall, Englewood Cliffs, New Jersey.
Swamy, M. N. S. , and K. Thulasiraman 1981. Graphs, Networks, and Algorithms, Wiley, New York.
Wilson, R. 1972. Introduction to Graph Theory, Academic Press, New York.
Wu, N. , and R. Coppins 1981. Linear Programming and Extensions, McGraw-Hill, New York.

Computer Representation and Solution

Baker, E. , and B. Golden , 1982. The Analysis of Network Optimization Algorithms: An Overview, Working Paper MS/S No. 82-022, College of Business and Management, University of Maryland.
Ball, M. , and M. Magazine , 1981. The Design and Analysis of Heuristics, Networks, 11, pp. 215219.
Brassard, G. , and P. Bratley 1988. Algorithmes, Theory and Practice, Prentice-Hall, Englewood Cliffs, New Jersey.
45 Fisher, M. , 1980. Worst-Case Analysis of Heuristic Algorithms, Man. Sei., 26, pp. 117.
Foulds, L. R. , 1983. The Heuristic Problem-Solving Approach, J. Oper. Res. Soc., 34, pp. 927934.
Fox, B. L. , 1978. Data Structures and Computer Science Techniques in Operations Research, Oper. Res., 26, pp. 686717.
Hu, T. C. 1982. Combinatorial Algorithms, Addison-Wesley, Reading, Massachusetts.
Johnson, D. S. , and C. H. Papadimitriou , 1985. Computational Complexity. In The Traveling Salesman Problem (eds. E. L. Lawler , J. K. Lenstra , A. H. G. Rinnooy Kan , and D. B. Shmoys), Wiley, London.
Muller-Merbach, H. , 1981. Heuristics and Their Design: A Survey, Eur. J. Oper. Res., 8, pp. 123.
Papadimitriou, C. H. , and K. Steiglitz 1982. Combinatorial Optimization, Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey.
Syslo, M. M. , N. Deo , and J. S. Kowalik 1983. Discrete Optimization Algorithms with PASCAL Programs, Prentice-Hall, Englewood Cliffs, New Jersey.
Zanakis, S. H. , J. R. Evans , and A. A. Vazacopoulos , 1989. Heuristic Methods and Applications: A Categorized Survey, Eur. J. Oper. Res., 43, pp. 88110.

Tree Algorithms

Chandy, K. M. , and T. Lo , 1973. The Capacitated Minimum Spanning Tree, Networks, 3, pp. 173181.
Edmonds, J. , 1968. Optimum Branchings. In Mathematics of the Decision Sciences (eds. G. Dantzig and A. Veinott). Lectures in Applied Mathematics, Vol. 2. American Mathematical Society, Providence, Rhode Island, pp. 346361.
Gabow, H. N. , 1978. A Good Algorithm for Smallest Spanning Trees with a Degree Constraint, Networks, 8, pp. 201208.
Glover, F. , and D. Kingman , 1974. Finding Minimum Spanning Trees with a Fixed Number of Links at a Node, Colloquia Mathematica Societatis Janos Bolyai 12. Progress in Operations Research, Eger, Hungary, pp. 425439.
Hu, T. C. , 1961. The Maximum Capacity Route Problem, Oper. Res., 9, pp. 898900.
Hu, T. C. , 1974. Optimum Communication Spanning Trees, SIAM J. Comput., 3, pp. 188195.
Kou, L. , G. Markowsky , and L. Berman , 1978. A Fast Algorithm for Steiner Trees, Research Report RC 7390, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.
Kruskal, J. B. , 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, Proc. AMS, 7, pp. 4850.
Ng, S. M. , 1989. A New Spanning Tree Algorithm for Group Technology Problems, Working Paper 1989-17, Industrial and Systems Engineering, University of Southern California, Los Angeles.
Pollock, M. , 1960. The Maximum Capacity Route Through a Network, Oper. Res., 8, pp. 733736.
Prim, R. C. 1957. Shortest Connection Networks and Some Generalizations, Bell Syst. Tech. J., 36, pp. 13891401.
Rosenstiehl, P. , 1967. LArbre Minimum dun Graphe, Proceedings International Symposium on the Theory of Graphs in Rome, 1966, Donod/Gordon-Breach, Rome.
Winter, P. , 1987. Steiner Problem in Networks: A Survey, Networks, 17, pp. 129167.

Shortest-Path Algorithms

Dantzig, G. B. , 1960. On the Shortest Route Through a Network, Man. Sci. 6, pp. 187190.
Dantzig, G. B. , 1967. All Shortest Routes in a Graph, Theory of Graphs, International Symposium, Rome, 1966, Gordon and Breach, New York, pp. 9192.
Deo, N. , and C. Pang , 1984. Shortest-Path Algorithms: Taxonomy and Annotation, Networks, 14, pp. 275323.
122 Dijkstra, E. W. , 1959. A Note on Two Problems in Connexion with Graphs, Numer. Math., I, pp. 269271.
Dreyfus, S. E. , 1969. An Appraisal of Some Shortest Path Algorithms, Oper. Res., 17, pp. 395412.
Floyd, R. W. , 1962. Algorithm 97, Shortest Path, Commun. ACM, 5, p. 345.
Ford, L. R. , 1956. Network Flow Theory, Rand Corporation Report, p. 923.
Gallo, G. , and S. Pallotino , 1986. Shortest Path Methods: A Unifying Approach, Math. Program. Study, 26, pp. 3864.
Glover, F. , D. Klingman , and N. Phillips , 1985a. A New Polynomially Bounded Shortest Path Algorithm, Oper. Res., 33, pp. 6573.
Glover, F. , D. Klingman , N. Phillips , and R. Schneider , 1985b. New Polynomial Shortest Path Algorithms and the Computational Attributes, Manage. Sci., 31, pp. 11061128.
Helgason, R. V. , J. L. Kennington , and B. D. Stewart , 1988. Dijkstras Two-Tree Shortest Path Algorithm, Technical Report 88-OR-13, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas; revised July 1988.
Hung, M. , and J. J. Divoky , 1988. A Computational Study of Efficient Shortest Path Algorithms, Comput. Oper. Res., 15, pp. 567576.
Karp, R. M. , 1975. On the Computational Complexity of Combinatorial Problems, Networks, 5, pp. 4568.
Knuth, D. E. 1973. The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Massachusetts.
Lawler, E. L. , 1971. The Complexity of Combinatorial Computations: A Survey, Proc. 1971 Polytechnic Institute of Brooklyn Symposium on Computers and Automata.

Minieka, E. T. , 1974. On Computing Sets of Shortest Paths in a Graph, Commun. ACM, 17, pp. 351353.
Minieka, E. T. , and D. R. Shier , 1973. A Note on an Algebra for the k Best Routes in a Network, J. IMA, 11, pp. 145149.
Nicholson, T. A. , 1966, Finding the Shortest Route Between Two Points in a Network, Comput. J., 9, pp. 275280.
Shier, D. R. , 1974. Computational Experience with an Algorithm for Finding the k Shortest Paths in a Network, J. Res. Natl. Bur. Stand., 784(JulySeptember), pp. 139165.
Shier, D. R. , 1976. Iterative Methods for Determining the k Shortest Paths in a Network, Networks, 6, pp. 2052320.
Shier, D. R. , 1979. On Algorithms for Finding the k Shortest Paths in a Network, Networks, 9, pp. 195214.
Williams, T. A. , and G. P. White , 1973. A Note on Yens Algorithms for Finding the Length of All Shortest Paths in N-Node Nonnegative Distance Networks, J. ACM, 20(3), pp. 389390.
Yen, J. Y. , 1970. An Algorithm for Finding Shortest Routes from All Source Nodes to a Given Destination in General Networks, Q. Appl. Math., 27, pp. 526530.
Yen, J. Y. , 1973. Finding the Lengths of All Shortest Paths in N-Node Nonnegative Distance Complete Networks Using 1/2N3 Additions and N3 Comparisons, J. ACM, 19(3), pp. 423424.

Minimum-Cost Flow Algorithms

Barr, R. S. , and J. S. Turner , 1981. Microdata File Merging Through Large Scale Network Technology. In Network Models and Associated Applications (eds. D. Klingman and J. M. Mulvey), North-Holland, Amsterdam, pp. 122.
Bazaraa, M. S. , J. J. Jarvis , and H. Sherali 1990. Linear Programming and Network Flows, 2nd ed., Wiley & Sons, New York.
Bowers, M. R. , and J. P. Jarvis , 1989. A Hierarchical Production Planning and Scheduling Model. Report 248, University of Tennessee, College of Business Administration, Knoxville.
Bradley, G. H. , G. G. Brown , and G. W. Graves , 1977. Design and Implementation of Large Scale Primal Transshipment Algorithms, Man. Sei., 24, pp. 134.
Brown, G. , and R. D. McBride , 1984. Solving Generalized Networks, Man. Sei., 30, pp. 14971523.
Busaker, R. G. , and P. J. Gowen , 1961. A Procedure for Determining a Family of Minimal-Cost Network Flow Patterns, O.R.O. Technical Report No. 15, Operational Research Office, Johns Hopkins University, Baltimore.
Cooper, R. B. 1972. Introduction to Queueing Theory, Macmillan, New York.
Grinold, R. C. , 1973. Calculating Maximal Flows in a Network with Positive Gains, Oper. Res., 21, pp. 528541.
Iri, M. A New Method of Solving Transportation-Network Problems, Journal of the Operations Research Society of Japan, 3, pp. 2787.
Jewell, W. S. , 1958. Optimal Flow Through Networks, Interim Technical Report No. 8, Operations Research Center, MIT, Cambridge, Massachusetts.
Jewell, W. S. , 1962. Optimal Flow Through a Network with Gains, Oper. Res., 10, pp. 476499.
Johnson, E. L. , 1966. Networks and Basic Solutions, Oper. Res., 14, pp. 619623.
Klein, M. , 1967. A Primal Method for Minimal Cost Flows, Man. Sei., 14, pp. 205220.
Larson, R. C. , and A. R. Odoni 1981. Urban Operations Research, Prentice-Hall, Englewood Cliffs, New Jersey.
Maurras, J.F. , 1972. Optimization of the Flow Through Networks with Gains, Math. Program., 3, pp. 135144.
Minieka, E. T. , 1972. Optimal Flow in a Network with Gains, INFOR, 10, pp. 171178.
Truemper, K. , 1973, Optimum Flow in Networks with Positive Gains, Ph.D. Thesis, Case Western Reserve University, Cleveland.
Truemper, K. , 1976. An Efficient Scaling Procedure for Gains Networks, Networks, 6, pp. 151160.

Maximum-Flow Algorithms

Aronson, J. E. , 1989. A Survey of Dynamic Network Flows, Ann. Oper. Res.
Edmonds, J. , and R. M. Karp , 1972. Theoretical Improvements in Algorithm Efficiency for Network Flow Problems, J. ACM, 19, no. 2, pp. 218264.
Ford, L. R. , and D. R. Fulkerson 1962. Flows in Networks, Princeton University Press, Princeton, New Jersey.
Grinold, R. C. , 1973. Calculating Maximal Flows in a Network with Positive Gains, Oper. Res., 21, pp. 528541.
Jewell, W. S. , 1962. Optimal Flow through a Network with Gains, Oper. Res., 10, pp. 476499.
Johnson, E. L. , 1966. Networks and Basic Solutions, Oper. Res., 14, pp. 619623.
Karzanov, A. V. , 1974. Determining the Maximal Flow in a Network by the Method of Preflows, Sov. Math. Dokl., 15, pp. 434437.
Maurras, J. F. , 1972. Optimization of the Flow through Networks with Gains, Math. Program., 3, pp. 135144.
Maxwell, W. L. , and R. C. Wilson , 1981. Dynamic Network Flow Modeling of Fixed Path Material Handling Systems, AHE Trans., 13, no. 1, pp. 1221.
Minieka, E. T. , 1973. Maximum, Lexicographic and Dynamic Network Flows, Oper. Res., 21, pp. 517527.
Minieka, E. T. , 1972. Optimal Flow in a Network with Gains, INFOR, 10, pp. 171178.
Truemper, K. , 1973. Optimum Flow in Networks with Positive Gains, Ph.D. Thesis, Case Western Reserve University, Cleveland.
Truemper, K. , 1976. An Efficient Scaling Procedure for Gains Networks, Networks, 6, pp. 151160.
Wilkinson, W. L. , 1971. An Algorithm for Universal Maximal Dynamic Flows in a Network, Oper. Res., 19, pp. 16021612.

Matching and Assignment Algorithms

Balinksi, M. , 1969. Labelling to Obtain a Maximum Matching. In Combinatorial Mathematics and Its Applications (eds. R. C. Bose and T. A. Dowling), University of North Carolina Press, Chapel Hill, pp. 585602.
Ball, B. C. , and D. B. Webster , 1977. Optimal Scheduling for Even-Numbered Team Athletic Conferences, AIIE Trans., 9, pp. 161169.
Berge, C. , 1957. Two Theorems in Graph Theory, Proc. Natl. Acad. Sci. U.S.A., 43, pp. 842844. Brown, J.R. (undated). Maximum Cardinality Matching, Kent State University, unpublished manuscript.
Campbell, R. T. , and D.S. Chen , 1976. A Minimum Distance Basketball Scheduling Problem. In Management Science in Sports (eds. R. Machol et al.), North-Holland, Amsterdam, pp. 1525.
Edmonds, J. , 1965. Paths, Trees, and Flowers, Can. J. Math., 17, pp. 449467.
Edmonds, J. , 1965. Maximum Matching and Polyhedra with 0-1 Vertices, J. Res. Natl. Bur. Stand. 695(1,2), pp. 125130.
277 Edmonds, J. , and E. Johnson , 1970. Matching: A Well Solved Class of Integer Linear Programs, Combinatorial Structures and Their Applications, Gordon and Breach, New York, pp. 8992.
Evans, James R. A Microcomputer-Based Decision Support System for Scheduling Umpires in the American Baseball League, Interfaces, Vol. 18, No. 6, November-December 1988, pp. 4251.
Jonker, R. , and T. Volgenant , 1986. Improving the Hungarian Assignment Algorithm, Oper. Res. Lett., 5, pp. 171175.
Kuhn, H. W. , 1955. The Hungarian Method for the Assignment Problem, Naval Research Logistics Quarterly, 1, pp. 8397.
White, L. J. , 1967. A Parametric Study of Matchings and Coverings in Weighted Graphs, Ph.D. Thesis, University of Michigan.

The Postman and Related Arc Routing Problems

Beltrami, E. , and L. Bodin , 1974. Networks and Vehicle Routing for Municipal Waste Collection, Networks, 4(1), pp. 6594.
316 Christofides, N. , 1973. The Optimum Traversal of a Graph, OMEGA: Int. J. Manage. Sci., 1(6), pp. 719732.

Edmonds, J. , and E. L. Johnson , 1973. Matching, Euler Tours and the Chinese Postman, Math. Prog., 5, pp. 88124. (This paper provides an excellent treatment of postman results, additional methods for generating Euler tours, and a long bibliography for this problem.)
Golden, B. , and R. T. Wong , 1981. Capacitated Arc Routing Problems, Networks, II, pp. 305315.
Golden, B. , J. S. DeArmon , and E. K. Baker , 1983. Computational Experiments with Algorithms for a Class of Routing Problems, Comput. Oper. Res., 10, pp. 4769.
Lin, Y. , and Y. Zhao , 1988. A New Algorithm for the Directed Chinese Postman Problem, Comput. Oper. Res., 15(6), pp. 577584.

The Traveling Salesman and Related Vertex Routing Problems

Bellmore, M. , and G. L. Nemhauser , 1968. The Traveling Salesman Problem: A Survey, Oper. Res., 16, pp. 538558.
Berge, C. 1973. Graphs and Hypergraphs (English translation by E. Minieka), North-Holland, Amsterdam, Chapter 10, pp. 186227.
Bodin, L. , and B. Golden , 1981. Classification in Vehicle Routing and Scheduling, Networks, 11, pp. 97108.
Christofides, N. , 1972. Bounds for the Traveling-Salesman Problem, Oper. Res., 20(6), pp. 10441056.
Christofides, N. , 1976. Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem, Management Sciences Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.
Chvtal, V. , 1972. On Hamilton Ideals, J. Combinatorial Theory Ser. B, 12(2), pp. 163168.
Clarke, G. and J. W. Wright , 1963. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, Oper. Res., 12, pp. 568581.
Cornuejols, G. , and G. L. Nemhauser , 1978. Tight Bounds for Christofides Traveling Salesman Heuristic, Math. Program., 14, pp. 116121.
Garfinkel, R. , and G. L. Nemhauser 1972. Integer Programming, Wiley, New York, pp. 354360.
Ghouila-Houri, A. , 1960. Une condition suffisante dexistence dun circuit hamiltonien, C. R. Acad. Sei., 251, pp. 494497.
361 Held, M. , and R. Karp , 1970. The Traveling-Salesman Problem and Minimum Spanning Trees, Oper. Res., 18, pp. 11381162.
Held, M. , and R. Karp , 1971. The Traveling-Salesman Problem and Minimum Spanning Trees. Part II. Math. Program., 1(1), pp. 625.
Lenstra, J. K. , and A. H. G. Rinnooy Kan , 1975. Some Simple Applications of the Traveling Salesman Problem, Oper. Res. Q., 26(4), pp. 717733.
Lin, S. , 1965. Computer Solutions of the Traveling Salesman Problem, Bell Syst. Tech. J., 44, pp. 22452269.
Lin, S. , and B. W. Kernighan , 1973. An Effective Heuristic Algorithm for the Traveling-Salesman Problem, Oper. Res., 21(2), pp. 498516.
Magirou, V. F. , 1986. The Efficient Drilling of Printed Circuit Boards, Interfaces, 16(4), pp. 1323.
Ratliff, H. D. , and A. S. Rosenthal , 1983. Order Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem, Oper. Res., 31(3), pp. 507521.
Rosenkrantz, D. J. , R. E. Stearns , and P. M. Lewis , 1974. Approximate Algorithms for the Traveling Salesman Problem, Proc. 15th Ann. IEEE Symp. on Switching and Automatic Theory, pp. 3342.
Steckhan, H. , 1970. A Theorem on Symmetric Traveling Salesman Problems, Oper. Res., 18, pp. 11631167.

Location Problems

Christofides, N. , and P. Viola , 1971. The Optimum Location of Multicentres of a Graph, Oper. Res. Quart., 22, pp. 4554.
389 Cornuejols, G. , M. L. Fisher , and G. L. Nemhauser , 1977. Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms, Manage. Sei., 23(8), pp. 789810.
Dearing, P. M. , 1985. Review of Recent DevelopmentsLocation Problems, Oper. Res. Lett., 4(3), pp. 9598.
Goldman, A. J. , 1969. Optimum Locations for Centers in a Network, Transp. Sei., 4, pp. 352360.
Hakimi, S. L. , 1964. Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph, Oper. Res., 12, pp. 450459.

Hakimi, S. L. , 1965. Optimum Distribution of Switching Centers in a Communications Network and Some Graph Theoretic Problems, ORSA, 13, pp. 462475.
Halpern, J. , and O. Maimon , 1982. Algorithms for the m-Center Problems: A Survey, Eur. J. Oper. Res., 10, pp. 9099.
Handler, G. , and P. Mirchandani 1979. Location on Networks: Theory and Algorithms, MIT Press, Cambridge, Massachusetts.
Levy, J. , 1967. An Extended Theorem for Location in a Network, Oper. Res. Quart., 18, pp. 433442.
Marsten, R. , 1972. An Algorithm for Finding Almost All of the Medians of a Network, DP #23, Center for Mathematical Studies in Economics and Management Science, Northwestern University, Evanston, Illinois, November.
Minieka, E. , 1970. The m-Center Problem, SIAM Rev., 12, pp. 138139.
Minieka, E. , 1977. The General Centers and Medians of a Graph, Oper. Res., 25, pp. 641650.
Minieka, E. , 1981. A Polynomial Time Algorithm for Finding the Absolute Center of a Network, Networks, 11, pp. 351355.
ReVelle, G. , and R. Swain , 1970. Central Facility Location, Geogr. Anal. 2(1), pp. 3042.
Tansel, B. C. , R. L. Francis , and T. J. Lowe , 1983a. Location on Networks: A Survey. Part I: The p-Center and p-Median Problems, Manage. Sei., 29(4), pp. 482497.
Tansel, B. C. , R. L. Francis , and T. J. Lowe , 1983b. Location on Networks: A Survey. Part II: Exploiting Tree Network Structure, Manage. Sei., 29(4), pp. 498511.
Toregas, C. , C. ReVelle , R. Swain , and L. Bergman , 1971. The Location of Emergency Facilities, Oper. Res., 19, pp. 13631373.
Wendell, R. E. , and A. P. Hurter, Jr. , 1973. Optimal Locations on a Network, Transp. Sei., 7, pp. 1833.

Project Networks

Eisner, H. , 1962. A Generalized Network Approach to the Planning and Scheduling of a Research Project, Oper. Res., 10, pp. 115125.
Elmaghraby, S. , 1964. An Algebra for the Analysis of Generalized Activity Networks, Manage. Sei., 10(3), pp. 494514.
Evans, J. R. , D. R. Anderson , D. J. Sweeney , and T. A. Williams 1990. Applied Production and Operations Management, 3rd ed., West, St. Paul.
Ford, L. R. , and D. R. Fulkerson 1962. Flows in Networks, Princeton University Press, Princeton, pp. 151161.
Kelley, J. E., Jr. , 1969. Critical-Path Planning and Scheduling: Mathematical Basis, Oper. Res., 9, pp. 296320.
Moder, J. , and C. Phillips 1970. Project Management with CPM arid PERT, 2nd ed., Van Nostrand Reinhold, New York, (This is an excellent comprehensive introductory treatment of project networks.)
Pritsker, A. B. 1977. Modeling and Analysis Using Q-GERT Networks, Halsted Press, New York.
Pritsker, A. B. , and W. W. Happ , 1966. GERT: Graphical Evaluation Review Technique; Part I. Fundamentals, J. Ind. Eng., 17, pp. 267274.
Pritsker, A. B. , and G. Whitehouse , 1966. GERT: Graphical Evaluation Review Technique; Part 11. Probabilistic and Industrial Engineering Applications, J. Ind. Eng., 17, pp. 293301.418