

Integer Programming

LAURENCE A. WOLSEY

WILEY-INTERSCIENCE
SERIES IN DISCRETE MATHEMATICS AND OPTIMIZATION

ADVISORY EDITORS

RONALD L. GRAHAM
AT & T Laboratories, Florham Park, New Jersey, U.S.A.

JAN KAREL LENSTRA
*Department of Mathematics and Computer Science,
Eindhoven University of Technology, Eindhoven, The Netherlands*

ROBERT E. TARJAN
*Princeton University, New Jersey, and
NEC Research Institute, Princeton, New Jersey, U.S.A.*

A complete list of titles in this series appears at the end of this volume.



A Wiley-Interscience Publication
JOHN WILEY & SONS, INC.

New York • Chichester • Weinheim • Brisbane • Singapore • Toronto

To Marguerite

This text is printed on acid-free paper. ©

Copyright © 1998 by John Wiley & Sons, Inc. All rights reserved.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, (212) 850-6011, fax (212) 850-6008, E-Mail: PERMREQ@WILEY.COM.

Library of Congress Cataloging-in-Publication Data:

Wolsey, Laurence A.

Integer programming / Laurence A. Wolsey

p. cm. — (Wiley-Interscience series in discrete mathematics and optimization)

"Wiley-Interscience publication."

Includes bibliographical references and indexes.

ISBN 0-471-28366-5 (alk. paper)

1. Integer programming. I. Title. II. Series.

T57.74W67 1998

519.77—dc21

98-7296

Printed in the United States of America

10 9 8

Contents

Preface	xiii
Abbreviations and Notation	xvii
1 Formulations	1
1.1 Introduction	1
1.2 What Is an Integer Program?	3
1.3 Formulating IPs and BIPs	5
1.4 The Combinatorial Explosion	8
1.5 Mixed Integer Formulations	9
1.6 Alternative Formulations	12
1.7 Good and Ideal Formulations	14
1.8 Notes	18
1.9 Exercises	19
2 Optimality, Relaxation, and Bounds	23
2.1 Optimality and Relaxation	23
2.2 Linear Programming Relaxations	25
2.3 Combinatorial Relaxations	26
2.4 Lagrangian Relaxation	27
2.5 Duality	28

2.6	Primal Bounds: Greedy and Local Search	30
2.7	Notes	33
2.8	Exercises	33
3	Well-Solved Problems	37
3.1	Properties of Easy Problems	37
3.2	IPs with Totally Unimodular Matrices	38
3.3	Minimum Cost Network Flows	40
3.4	Special Minimum Cost Flows	42
3.4.1	Shortest Path	42
3.4.2	Maximum $s-t$ Flow	43
3.5	Optimal Trees	43
3.6	Submodularity and Matroids*	46
3.7	Notes	49
3.8	Exercises	50
4	Matchings and Assignments	53
4.1	Augmenting Paths and Optimality	53
4.2	Bipartite Maximum Cardinality Matching	55
4.3	The Assignment Problem	57
4.4	Notes	62
4.5	Exercises	63
5	Dynamic Programming	67
5.1	Some Motivation: Shortest Paths	67
5.2	Uncapacitated Lot-Sizing	68
5.3	An Optimal Subtree of a Tree	71
5.4	Knapsack Problems	72
5.4.1	0-1 Knapsack	73
5.4.2	Integer Knapsack Problems	74
5.5	Notes	77
5.6	Exercises	78
6	Complexity and Problem Reductions	81
6.1	Complexity	81
6.2	Decision Problems, and Classes \mathcal{NP} and \mathcal{P}	82
6.3	Polynomial Reduction and the Class \mathcal{NPC}	84
6.4	Consequences of $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$	87

6.5	Optimization and Separation	88
6.6	Notes	89
6.7	Exercises	89
7	Branch and Bound	91
7.1	Divide and Conquer	91
7.2	Implicit Enumeration	92
7.3	Branch and Bound: An Example	95
7.4	LP-based Branch and Bound	98
7.5	Using a Branch-and-Bound System	101
7.5.1	If All Else Fails	103
7.6	Preprocessing*	103
7.7	Notes	107
7.8	Exercises	108
8	Cutting Plane Algorithms	113
8.1	Introduction	113
8.2	Some Simple Valid Inequalities	114
8.3	Valid Inequalities	117
8.3.1	Valid Inequalities for Linear Programs	117
8.3.2	Valid Inequalities for Integer Programs	118
8.4	A Priori Addition of Constraints	121
8.5	Automatic Reformulation or Cutting Plane Algorithms	123
8.6	Gomory's Fractional Cutting Plane Algorithm	124
8.7	Mixed Integer Cuts	127
8.7.1	The Basic Mixed Integer Inequality	127
8.7.2	The Mixed Integer Rounding (MIR) Inequality	129
8.7.3	The Gomory Mixed Integer Cut*	129
8.8	Disjunctive Inequalities*	130
8.9	Notes	133
8.10	Exercises	134
9	Strong Valid Inequalities	139
9.1	Introduction	139
9.2	Strong Inequalities	140
9.2.1	Dominance	140
9.2.2	Polyhedra, Faces, and Facets	142

9.2.3	Facet and Convex Hull Proofs*	144
9.3	0-1 Knapsack Inequalities	147
9.3.1	Cover Inequalities	147
9.3.2	Strengthening Cover Inequalities	148
9.3.3	Separation for Cover Inequalities	150
9.4	Mixed 0-1 Inequalities	151
9.4.1	Flow Cover Inequalities	151
9.4.2	Separation for Flow Cover Inequalities	153
9.5	The Optimal Subtour Problem	154
9.5.1	Separation for Generalized Subtour Constraints	155
9.6	Branch-and-Cut	157
9.7	Notes	160
9.8	Exercises	161
10	Lagrangian Duality	167
10.1	Lagrangian Relaxation	167
10.2	The Strength of the Lagrangian Dual	172
10.3	Solving the Lagrangian Dual	173
10.4	Lagrangian Heuristics and Variable Fixing	177
10.5	Choosing a Lagrangian Dual	179
10.6	Notes	180
10.7	Exercises	181
11	Column Generation Algorithms	185
11.1	Introduction	185
11.2	Dantzig-Wolfe Reformulation of an IP	187
11.3	Solving the Master Linear Program	188
11.3.1	STSP by Column Generation	190
11.3.2	Strength of the Linear Programming Master	192
11.4	IP Column Generation for 0-1 IP	193
11.5	Implicit Partitioning/Packing Problems	194
11.6	Partitioning with Identical Subsets*	196
11.7	Notes	200
11.8	Exercises	201
12	Heuristic Algorithms	203
12.1	Introduction	203

12.2	Greedy and Local Search Revisited	204
12.3	Improved Local Search Heuristics	207
12.3.1	Tabu Search	207
12.3.2	Simulated Annealing	208
12.3.3	Genetic Algorithms	210
12.4	Worst-Case Analysis of Heuristics	211
12.5	MIP-based Heuristics	214
12.6	Notes	217
12.7	Exercises	218
13	From Theory to Solutions	221
13.1	Introduction	221
13.2	Software for Solving Integer Programs	221
13.3	How Do We Find an Improved Formulation?	223
13.3.1	Uncapacitated Lot-Sizing	223
13.3.2	Capacitated Lot-Sizing	227
13.4	Fixed Charge Networks: Reformulations	229
13.4.1	The Single Source Fixed Charge Network Flow Problem	229
13.4.2	The Directed Subtree Problem	231
13.5	Multi-Item Single Machine Lot-Sizing	232
13.6	A Multiplexer Assignment Problem	236
13.7	Notes	240
13.8	Exercises	241
	References	245
	Index	261

Preface

Intended Audience

The book is addressed to undergraduates and graduates in operations research, mathematics, engineering, and computer science. It should be suitable for advanced undergraduate and Masters level programs. It is also aimed at users of integer programming who wish to understand why some problems are difficult to solve, how they can be reformulated so as to give better results, and how mixed integer programming systems can be used more effectively.

The book is essentially self-contained, though some familiarity with linear programming is assumed, and a few basic concepts from graph theory are used.

The book provides material for a one-semester course of 2-3 hours per week.

What and How

Integer Programming is about ways to solve optimization problems with discrete or integer variables. Such variables are used to model indivisibilities, and 0/1 variables are used to represent on/off decisions to buy, invest, hire, and so on. Such problems arise in all walks of life, whether in developing train or aircraft timetables, planning the work schedule of a production line or a maintenance team, or planning nationally or regionally the daily or weekly production of electricity.

The last ten years have seen a remarkable advance in our ability to solve to near optimality difficult practical integer programs. This is due to a combination of

- i) Improved modeling
- ii) Superior linear programming software
- iii) Faster computers
- iv) New cutting plane theory and algorithms
- v) New heuristic methods
- vi) Branch-and-cut and integer programming decomposition algorithms

Today many industrial users still build an integer programming model and stop at the first integer feasible solution provided by their software. Unless the problem is very easy, such solutions can be 5%, 10%, or 100% away from optimal, resulting in losses running into mega-dollars. In many cases it is now possible to obtain solutions that are proved to be optimal, or proven within 0.1%, 1%, or 5% of optimal, in a reasonable amount of computer time. There is, however, a cost: better models must be built, and either specially tailored algorithms must be constructed, or better use must be made of existing commercial software.

To make such gains, it is necessary to understand why some problems are more difficult than others, why some formulations are better than others, how effective different algorithms can be, and how integer programming software can be best used. The aim of this book is to provide some such understanding.

Chapter 1 introduces the reader to various integer programming problems and their formulation, and introduces the important distinction between good and bad formulations. Chapter 2 explains how it is possible to prove that feasible solutions are optimal or close to optimal.

Chapters 3–5 study integer programs that are easy. The problems and algorithms are interesting in their own right, but also because the algorithmic ideas can often be adapted so as to provide good feasible solutions for more difficult problems. In addition, these easy problems must often be solved repeatedly as subproblems in algorithms for the more difficult problems. We examine when linear programs automatically have integer solutions, which is in particular the case for network flow problems. The greedy algorithm for finding an optimal tree, the primal-dual algorithm for the assignment problem, and a variety of dynamic programming algorithms are presented, and their running times examined.

In Chapter 6 we informally address the question of the apparent difference in difficulty between the problems presented in Chapters 3–5 that can be solved rapidly, and the “difficult” problems treated in the rest of the book.

The fundamental branch-and-bound approach is presented in Chapter 7. Certain features of commercial integer programming systems based on branch-and-bound are discussed. In Chapters 8 and 9 we discuss valid inequalities and cutting planes. The use of inequalities to improve formulations and obtain tighter bounds is the area in which probably the most progress has been made in the last ten years. We give examples of the cuts, and also the routines to find cuts that are being added to the latest systems.

In Chapters 10 and 11 two important ways of decomposing integer programs are presented. The first is by Lagrangian relaxation and the second by column generation. It is often very easy to implement a special-purpose algorithm based on Lagrangian relaxation, and many applications are reported in the literature. Integer programming column generation, which is linear programming based, is more recent, but several recent applications suggest that its importance will grow.

Whereas the emphasis in Chapters 7–11 is on obtaining “dual” bounds (upper bounds on the optimal value of a maximization problem), the need to find good feasible solutions that provide “primal” (lower) bounds is addressed in Chapter 12. We present the basic ideas of various modern local search metaheuristics, introduce briefly the worst-case analysis of heuristics, and also discuss how an integer programming system can be used heuristically to find solutions of reasonable quality for highly intractible integer programs.

Finally, in Chapter 13 we change emphasis. By looking at a couple of applications and asking a series of typical questions, we try to give a better idea of how theory and practice converge when confronted with the choice of an appropriate algorithm, and the question of how to improve a formulation, or how to use a commercial mixed integer programming system effectively.

In using the book for a one-semester course, the chapters can be taken in order. In any case we suggest that the basics consisting of Chapters 1, 2, 3, 6, 7 should be studied in sequence. Chapter 4 is interesting for those who have had little exposure to combinatorial optimization. Chapter 5 can be postponed, and parts of Chapter 12 can be studied at any time after Chapter 2. There is also no difficulty in studying Chapter 10 before Chapters 8 and 9. The longer Chapters 7, 8, 9 and 11 contain starred sections that are optional. The instructor may wish to leave out some material from these chapters, or alternatively devote more time to them. Chapter 13 draws on material from most of the book, but can be used as motivation much earlier.

Acknowledgments

I am sincerely grateful to the many people who have contributed in some way to the preparation of this book. Marko Loparic has voluntarily played the

role of teaching assistant in the course for which this material was developed. Michele Conforti, Cid de Souza, Eric Gourdin and Abilio Lucena have all used parts of it in the classroom and provided feedback. John Beasley, Marc Pirlot, Yves Pochet, James Tebboth and François Vanderbeck have criticized one or more chapters in detail, and Jan-Karel Lenstra has both encouraged and provided rapid feedback when requested. Finishing always takes longer than expected, and I am grateful to my colleagues and doctoral students at Core for their patience when they have tried to interest me in other more pressing matters, to the Computer Science Department of the University of Utrecht for allowing me to finish off the book in congenial and interesting surroundings, and to Esprit program 20118, MEMIPS, for support during part of the 1997–98 academic year. Sincere thanks go to Fabienne Henry for her secretarial help over many years, and for her work in producing the final manuscript.

Scientifically I am deeply indebted to the many researchers with whom I have had the good fortune and pleasure to collaborate. Working with George Nemhauser for many years has, I hope, taught me a little about writing. His earlier book on integer programming with R. Garfinkel provided an outstanding model for an undergraduate textbook. Yves Pochet is a considerate and stimulating colleague, and together with Bob Daniel, who has always been ready to provide a “practical problem per day,” they provide a constant reminder that integer programming is challenging both theoretically and practically. However, the bias and faults that remain are entirely my own.

Abbreviations and Notation

BIP: Binary or Zero-One Integer Problem
 B^n : $\{0, 1\}^n$ the set of n -dimensional 0,1 vectors
C-G: Chvátal-Gomory
CLS: Capacitated Lot-Sizing Problem
 $\text{conv}(S)$: The convex hull of S
COP: Combinatorial Optimization Problem
D: Dual Problem
DP: Dynamic Programming
 e_j : The j^{th} unit vector
 e^S : The characteristic vector of S
 $E(S)$: All edges with both endpoints in the node set S
FCNF: Fixed Charge Network Flow Problem
GAP: Generalized Assignment Problem
GSEC: Generalized Subtour Elimination Constraint
GUB: Generalized Upper Bound
IKP: Integer Knapsack Problem
IP: Integer Programming Problem
IPM: Integer Programming Master Problem
LD: Lagrangian Dual Problem
 lhs: left-hand side
LP: abbreviation for “linear programming”

- LP*: A specific or general Linear Programming Problem
- LPM*: Linear Programming Master Problem
- $L(X)$: Length of the input of a problem instance X
- M : A large positive number
- MIP*: Mixed Integer Programming Problem
- MIR*: Mixed Integer Rounding
- N : Generic set $\{1, 2, \dots, n\}$
- NP*: Class of NP problems
- NPC*: Class of NP-complete problems
- P : Generic Problem Class, or Polyhedron
- \mathcal{P} : Class of polynomially solvable problems
- $\mathcal{P}(N)$: Set of subsets of N
- rhs: right-hand side
- RLPM*: Restricted Linear Programming Master Problem
- R^n : The n -dimensional real numbers
- R_+^n : The n -dimensional nonnegative real numbers
- S : Feasible region of *IP*, or subset of N
- SOS*: Special Ordered Set
- STSP*: Symmetric Traveling Salesman Problem
- TSP*: (Asymmetric) Traveling Salesman Problem
- TU*: Totally Unimodular
- UFL*: Uncapacitated Facility Location Problem
- ULS*: Uncapacitated Lot-Sizing Problem
- $V^-(i)$: Node set $\{k \in V : (k, i) \in A\}$
- $V^+(i)$: Node set $\{k \in V : (i, k) \in A\}$
- X : Feasible region of *IP*, or a problem instance
- $(x)^+$: The maximum of x and 0
- Z_+^n : The n -dimensional nonnegative integers
- $\delta^-(S)$: All arcs going from a node not in S to a node in S
- $\delta^+(S)$: All arcs going from a node in S to a node not in S
- $\delta(S)$ or $\delta(S, V \setminus S)$: All edges with one endpoint in S and the other in $V \setminus S$
- $\delta(i)$ or $\delta(\{i\})$: The set of edges incident to node i
- $\mathbf{1}$: The vector $(1, 1, \dots, 1)$
- $\mathbf{0}$: The vector $(0, 0, \dots, 0)$

Integer Programming

1

Formulations

1.1 INTRODUCTION

A wide variety of practical problems can be formulated and solved using integer programming. We start by describing briefly a few such problems.

1. Train Scheduling

Certain train schedules repeat every hour. For each line, the travel times between stations are known, and the time spent in a station must lie within a given time interval. Two trains traveling on the same line must for obvious reasons be separated by at least a given number of minutes. To make a connection between trains A and B at a particular station, the difference between the arrival time of A and the departure time of B must be sufficiently long to allow passengers to change, but sufficiently short so that the waiting time is not excessive. The problem is to find a feasible schedule.

2. Airline Crew Scheduling

Given the schedule of flights for a particular aircraft type, one problem is to design weekly schedules for the crews. Each day a crew must be assigned a duty period consisting of a set of one or more linking flights satisfying numerous constraints such as limited total flying time, minimum rests between flights, and so on. Then putting together the duty periods, weekly schedules or pairings are constructed which must satisfy further constraints on overnight rests, flying time, returning the crew to its starting point, and so on. The objective is to minimize the amount paid to the crews, which is a function of flying time, length of the duty periods and pairings, a guaranteed minimum

number of flying hours, and so forth.

3. Production Planning

A multinational company holds a monthly planning meeting in which a three-month production and shipping plan is drawn up based on their latest estimates of potential sales. The plan covers 200–400 products produced in 5 different factories with shipments to 50 sales areas. Solutions must be generated on the spot, so only about 15 minutes' computation time is available. For each product there is a minimum production quantity, and production is in batches — multiples of some fixed amount. The goal is to maximize contribution.

4. Electricity Generation Planning

A universal problem is the unit commitment problem of developing an hourly schedule spanning a day or a week so as to decide which generators will be producing and at what levels. Constraints to be satisfied include satisfaction of estimated hourly or half-hourly demand, reserve constraints to ensure that the capacity of the active generators is sufficient should there be a sudden peak in demand, and ramping constraints to ensure that the rate of change of the output of a generator is not excessive. Generators have minimum on- and off-times, and their start-up costs are a nonlinear function of the time they have been idle.

5. Telecommunications

A typical problem given the explosion of demand in this area concerns the installation of new capacity so as to satisfy a predicted demand for data/voice transmission. Given estimates of the requirements between different centers, the existing capacity, and the costs of installing new capacity which is only available in discrete amounts, the problem is to minimize cost taking into account the possibility of failure of a line or a center due to a breakdown or accident.

6. Buses for the Handicapped (or Dial-a-Ride)

In several major cities a service is available whereby handicapped subscribers can call in several hours beforehand with a request to be taken from A to B at a certain time, with special facilities such as space for a wheel chair if necessary. The short-term problem is to schedule the fleet of specialized mini-buses so as to satisfy a maximum number of requests. One long-term problem is to decide the optimal size of the fleet.

7. Ground Holding of Aircraft

Given several airports, a list of flights, and the capacity of the airports in each period, which is a function of the weather conditions and forecasts, the problem is to decide which planes to delay and by how long, taking into account the numbers of passengers, connecting flights, the expected time until

conditions improve, and so on, with the objective of minimizing aircraft costs and passenger inconvenience.

8. Cutting Problems

Whether cutting lengths of paper from rolls, plastic from large rectangular sheets, or patterns to make clothes, the problem is in each case to follow precisely determined cutting rules, satisfy demand, and minimize waste.

Other recent application areas include problems in molecular biology, statistics, and VLSI.

This book tries to provide some of the understanding and tools necessary for tackling such problems.

1.2 WHAT IS AN INTEGER PROGRAM?

Suppose that we have a linear program

$$\max\{cx : Ax \leq b, x \geq 0\}$$

where A is an m by n matrix, c an n -dimensional row vector, b an m -dimensional column vector, and x an n -dimensional column vector of variables or unknowns. Now we add in the restriction that certain variables must take integer values.

If some but not all variables are integer, we have a **(Linear) Mixed Integer Program**, written as

$$(MIP) \quad \begin{array}{l} \max cx + hy \\ Ax + Gy \leq b \\ x \geq 0, y \geq 0 \text{ and integer} \end{array}$$

where A is again m by n , G is m by p , h is a p row-vector, and y is a p column-vector of integer variables.

If all variables are integer, we have a **(Linear) Integer Program**, written as

$$(IP) \quad \begin{array}{l} \max cx \\ Ax \leq b \\ x \geq 0 \text{ and integer,} \end{array}$$

and if all variables are restricted to 0–1 values, we have a **0–1 or Binary Integer Program**

$$(BIP) \quad \begin{array}{l} \max cx \\ Ax \leq b \\ x \in \{0, 1\}^n. \end{array}$$

Another type of problem that we wish to study is a "combinatorial optimization problem." Here typically we are given a finite set $N = \{1, \dots, n\}$, weights c_j for each $j \in N$, and a set \mathcal{F} of feasible subsets of N . The problem of finding a minimum weight feasible subset is a **Combinatorial Optimization Problem**

$$(COP) \quad \min_{S \subseteq N} \left\{ \sum_{j \in S} c_j : S \in \mathcal{F} \right\}.$$

In the next section we will see various examples of *IPs* and *COPs*, and also see that often a *COP* can be formulated as an *IP* or *BIP*.

Given that integer programs look very much like linear programs, it is not surprising that linear programming theory and practice is fundamental in understanding and solving integer programs. However, the first idea that springs to mind, namely "rounding", is often insufficient, as the following example shows.

Example 1.1 Consider the integer program:

$$\begin{aligned} \max & 1.00x_1 + 0.64x_2 \\ & 50x_1 + 31x_2 \leq 250 \\ & 3x_1 - 2x_2 \geq -4 \\ & x_1, x_2 \geq 0 \text{ and integer.} \end{aligned}$$

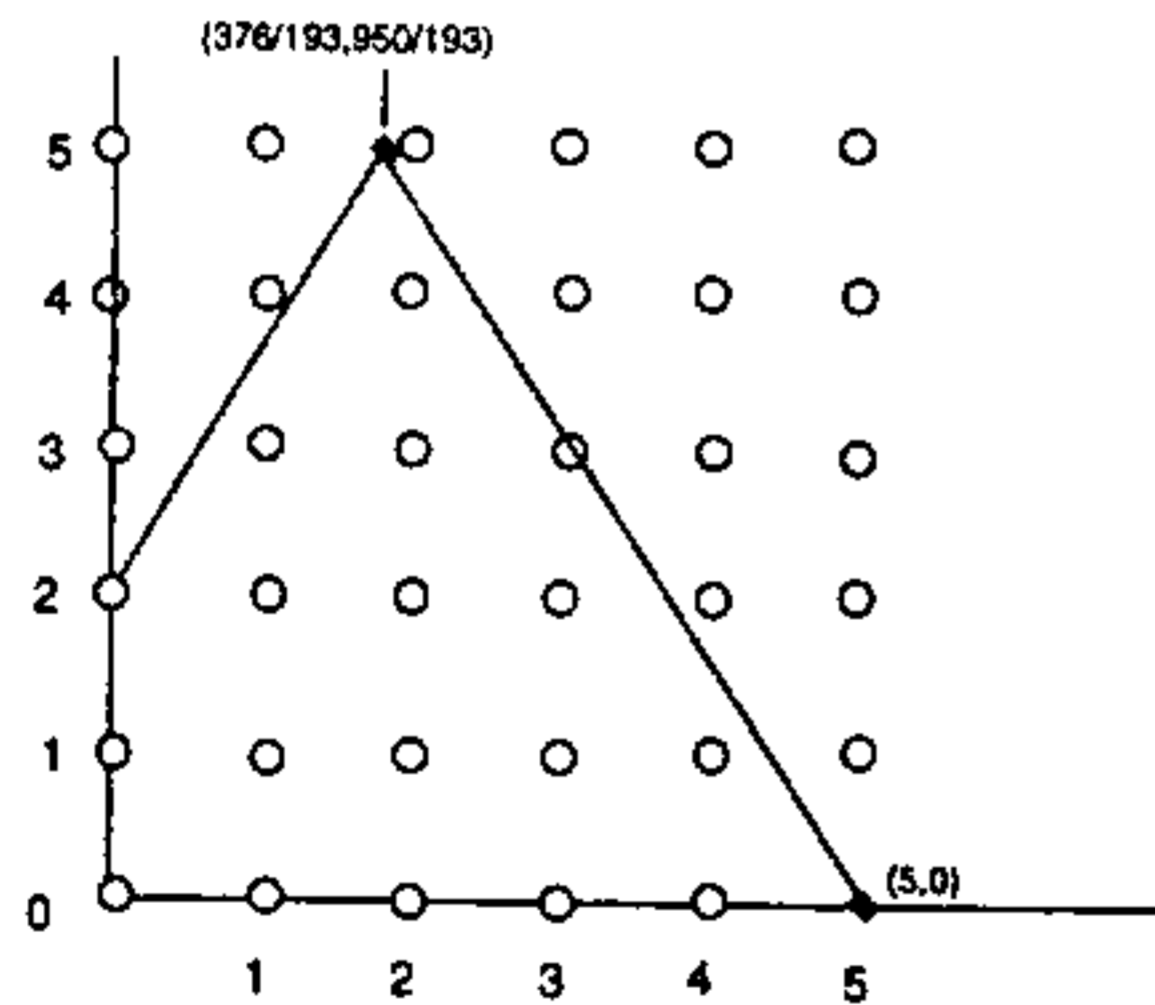


Fig. 1.1 Rounding the LP

As we see from Figure 1.1, the linear programming solution $(376/193, 950/193)$ is a long way from the optimal integer solution $(5, 0)$.

For *BIPs* the situation is often even worse. The linear programming solution may well be $(0.5, \dots, 0.5)$, giving no information whatsoever. What is more, it is typically very difficult just to answer the question whether there exists a feasible 0-1 solution.

1.3 FORMULATING IPS AND BIPS

As in linear programming, translating a problem description into a formulation should be done systematically, and a clear distinction should be made between the data of the problem instance, and the variables (or unknowns) used in the model.

- (i) Define what appear to be the necessary variables.
- (ii) Use these variables to define a set of constraints so that the feasible points correspond to the feasible solutions of the problem.
- (iii) Use these variables to define the objective function.

If difficulties arise, define an additional or alternative set of variables and iterate.

Defining variables and constraints may not always be as easy as in linear programming. Especially for *COPs*, we are often interested in choosing a subset $S \subseteq N$. For this we typically make use of the *incidence vector* of S , which is the n -dimensional 0-1 vector x^S such that $x_j^S = 1$ if $j \in S$, and $x_j^S = 0$ otherwise.

Below we formulate four well-known integer programming problems.

The Assignment Problem

There are n people available to carry out n jobs. Each person is assigned to carry out exactly one job. Some individuals are better suited to particular jobs than others, so there is an estimated cost c_{ij} if person i is assigned to job j . The problem is to find a minimum cost assignment.

Definition of the variables.

$x_{ij} = 1$ if person i does job j , and $x_{ij} = 0$ otherwise.

Definition of the constraints.

Each person i does one job:

$$\sum_{j=1}^n x_{ij} = 1 \text{ for } i = 1, \dots, n.$$

Each job j is done by one person:

$$\sum_{i=1}^n x_{ij} = 1 \text{ for } j = 1, \dots, n.$$

The variables are 0-1:

$$x_{ij} \in \{0, 1\} \text{ for } i = 1, \dots, n, j = 1, \dots, n.$$

Definition of the objective function.

The cost of the assignment is minimized:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}.$$

The 0-1 Knapsack Problem

There is a budget b available for investment in projects during the coming year and n projects are under consideration, where a_j is the outlay for project j , and c_j is its expected return. The goal is to choose a set of projects so that the budget is not exceeded and the expected return is maximized.

Definition of the variables.

$x_j = 1$ if project j is selected, and $x_j = 0$ otherwise.

Definition of the constraints.

The budget cannot be exceeded:

$$\sum_{j=1}^n a_j x_j \leq b.$$

The variables are 0-1:

$$x_j \in \{0, 1\} \text{ for } j = 1, \dots, n.$$

Definition of the objective function.

The expected return is maximized:

$$\max \sum_{j=1}^n c_j x_j.$$

The Set Covering Problem

Given a certain number of regions, the problem is to decide where to install a set of emergency service centers. For each possible center the cost of installing a service center, and which regions it can service are known. For instance, if the centers are fire stations, a station can service those regions for which a fire engine is guaranteed to arrive on the scene of a fire within 8 minutes. The goal is to choose a minimum cost set of service centers so that each region is covered.

First we can formulate it as a more abstract *COP*. Let $M = \{1, \dots, m\}$ be the set of regions, and $N = \{1, \dots, n\}$ the set of potential centers. Let $S_j \subseteq M$ be the regions that can be serviced by a center at $j \in N$, and c_j its installation cost. We obtain the problem:

$$\min_{T \subseteq N} \left\{ \sum_{j \in T} c_j : \cup_{j \in T} S_j = M \right\}.$$

Now we formulate it as a *BIP*. To facilitate the description, we first construct a 0-1 *incidence matrix* A such that $a_{ij} = 1$ if $i \in S_j$, and $a_{ij} = 0$ otherwise. Note that this is nothing but processing of the data.

Definition of the variables.

$x_j = 1$ if center j is selected, and $x_j = 0$ otherwise.

Definition of the constraints.

At least one center must service region i :

$$\sum_{j=1}^n a_{ij} x_j \geq 1 \text{ for } i = 1, \dots, m.$$

The variables are 0-1:

$$x_j \in \{0, 1\} \text{ for } j = 1, \dots, n.$$

Definition of the objective function.

The total cost is minimized:

$$\min \sum_{j=1}^n c_j x_j.$$

The Traveling Salesman Problem (*TSP*)

This is perhaps the most notorious problem in Operations Research because it is so easy to explain, and so tempting to try and solve. A salesman must visit each of n cities exactly once and then return to his starting point. The time taken to travel from city i to city j is c_{ij} . Find the order in which he should make his tour so as to finish as quickly as possible.

This problem arises in a multitude of forms: a truck driver has a list of clients he must visit on a given day, or a machine must place modules on printed circuit boards, or a stacker crane must pick up and depose crates. Now we formulate it as a *BIP*.

Definition of the variables.

$x_{ij} = 1$ if the salesman goes directly from town i to town j , and $x_{ij} = 0$ otherwise. (x_{ii} is not defined for $i = 1, \dots, n$.)

Definition of the constraints.

He leaves town i exactly once:

$$\sum_{j:j \neq i} x_{ij} = 1 \text{ for } i = 1, \dots, n.$$

He arrives at town j exactly once:

$$\sum_{i:i \neq j} x_{ij} = 1 \text{ for } j = 1, \dots, n.$$

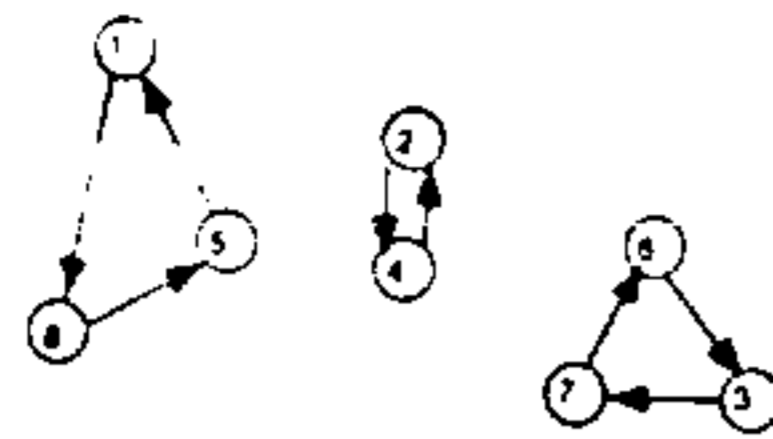


Fig. 1.2 Subtours

So far these are precisely the constraints of the assignment problem. A solution to the assignment problem might give a solution of the form shown in Figure 1.2 (i.e., a set of disconnected subtours). To eliminate these solutions, we need more constraints that guarantee connectivity by imposing that the salesman must pass from one set of cities to another, so-called *cut-set* constraints:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \text{ for } S \subset N, S \neq \phi.$$

An alternative is to replace these constraints by *subtour elimination* constraints:

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \text{ for } S \subset N, 2 \leq |S| \leq n - 1.$$

The variables are 0-1:

$$x_{ij} \in \{0, 1\} \text{ for } i = 1, \dots, n, j = 1, \dots, n, i \neq j.$$

Definition of the objective function.
The total travel time is minimized:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}.$$

1.4 THE COMBINATORIAL EXPLOSION

The four problems we have looked at so far are all combinatorial in the sense that the optimal solution is some subset of a finite set. Thus in principle these problems can be solved by enumeration. To see for what size of problem instances this is a feasible approach, we need to count the number of possible solutions.

The Assignment Problem. There is a one-to-one correspondence between assignments and permutations of $\{1, \dots, n\}$. Thus there are $n!$ solutions to

compare.

The Knapsack and Covering Problems. In both cases the number of subsets is 2^n . For the knapsack problem with $b = \sum_{j=1}^n a_j/2$, at least half of the subsets are feasible, and thus there are at least 2^{n-1} feasible subsets.

The Traveling Salesman Problem. Starting at city 1, the salesman has $n - 1$ choices. For the next choice $n - 2$ cities are possible, and so on. Thus there are $(n - 1)!$ feasible tours.

In Table 1.1 we show how rapidly certain functions grow. Thus a *TSP* with $n = 101$ has approximately 9.33×10^{157} tours.

n	$\log n$	$n^{0.5}$	n^2	2^n	$n!$
10	3.32	3.16	10^2	1.02×10^3	3.6×10^6
100	6.64	10.00	10^4	1.27×10^{30}	9.33×10^{157}
1000	9.97	31.62	10^6	1.07×10^{301}	4.02×10^{2567}

Table 1.1 Some typical functions

The conclusion to be drawn is that using complete enumeration we can only hope to solve such problems for very small values of n . Therefore we have to devise some more intelligent algorithms, otherwise the reader can throw this book out of the window.

1.5 MIXED INTEGER FORMULATIONS

Modeling Fixed Costs

Suppose we wish to model a typical nonlinear fixed charge cost function:

$$h(x) = f + px \text{ if } 0 < x \leq C \text{ and } h(x) = 0 \text{ if } x = 0$$

with $f > 0$ and $p > 0$ (see Figure 1.3).

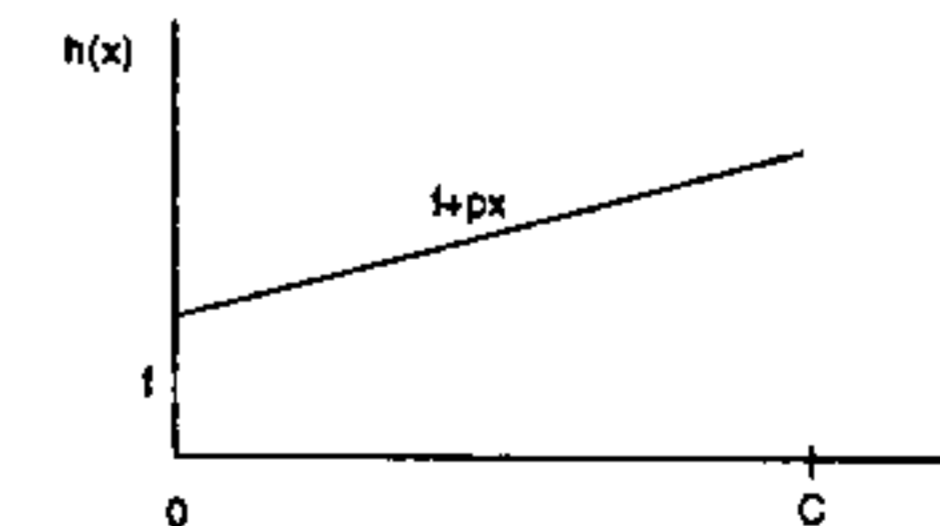


Fig. 1.3 Fixed cost function

Definition of an additional variable.

$$y = 1 \text{ if } x > 0 \text{ and } y = 0 \text{ otherwise.}$$

Definition of the constraints and objective function.

We replace $h(x)$ by $fy + px$, and add the constraints $x \leq Cy, y \in \{0, 1\}$.

Note that this is not a completely satisfactory formulation, because although the costs are correct when $x > 0$, it is possible to have the solution $x = 0, y = 1$. However, as the objective is minimization, this will typically not arise in an optimal solution.

Uncapacitated Facility Location (UFL)

Given a set of potential depots $N = \{1, \dots, n\}$ and a set $M = \{1, \dots, m\}$ of clients, suppose there is a fixed cost f_j associated with the use of depot j , and a transportation cost c_{ij} if all of client i 's order is delivered from depot j . The problem is to decide which depots to open, and which depot serves each client so as to minimize the sum of the fixed and transportation costs. Note that this problem is similar to the covering problem, except for the addition of the variable transportation costs.

Definition of the variables.

We introduce a fixed cost or depot opening variable $y_j = 1$ if depot j is used, and $y_j = 0$ otherwise.

x_{ij} is the fraction of the demand of client i satisfied from depot j .

Definition of the constraints.

Satisfaction of the demand of client i :

$$\sum_{j=1}^n x_{ij} = 1 \text{ for } i = 1, \dots, m.$$

To represent the link between the x_{ij} and the y_j variables, we note that $\sum_{i \in M} x_{ij} \leq m y_j$, and use the fixed cost formulation above to obtain:

$$\sum_{i \in M} x_{ij} \leq m y_j \text{ for } j \in N, x_{ij} \geq 0 \text{ for } i \in M, j \in N, y_j \in \{0, 1\} \text{ for } j \in N.$$

Definition of the objective function.

The objective is $\sum_{j \in N} h_j(x_{1j}, \dots, x_{mj})$ where $h_j(x_{1j}, \dots, x_{mj}) = f_j + \sum_{i \in M} c_{ij} x_{ij}$ if $\sum_{i \in M} x_{ij} > 0$, so we obtain

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{j \in N} f_j y_j.$$

Uncapacitated Lot-Sizing (ULS)

The problem is to decide on a production plan for an n -period horizon for a single product. The basic model can be viewed as having data:

f_t is the fixed cost of producing in period t .

p_t is the unit production cost in period t .

h_t is the unit storage cost in period t .

d_t is the demand in period t .

We use the natural (or obvious) variables:

x_t is the amount produced in period t .

s_t is the stock at the end of period t .

$y_t = 1$ if production occurs in t , and $y_t = 0$ otherwise.

To handle the fixed costs, we observe that a priori no upper bound is given on x_t . Thus we either must use a very large value $C = M$, or calculate an upper bound based on the problem data.

For constraints and objective we obtain:

$$\begin{aligned} \min \sum_{t=1}^n p_t x_t + \sum_{t=1}^n h_t s_t + \sum_{t=1}^n f_t y_t \\ s_{t-1} + x_t = d_t + s_t \text{ for } t = 1, \dots, n \\ x_t \leq M y_t \text{ for } t = 1, \dots, n \\ s_0 = 0, s_t, x_t \geq 0, y_t \in \{0, 1\} \text{ for } t = 1, \dots, n. \end{aligned}$$

If we impose that $s_n = 0$, then we can tighten the variable upper bound constraints to $x_t \leq (\sum_{i=t}^n d_i) y_t$. Note also that by substituting $s_t = \sum_{i=1}^t x_i - \sum_{i=1}^t d_i$, the objective function can be rewritten as $\sum_{t=1}^n c_t x_t + \sum_{t=1}^n f_t y_t - K$ where $c_t = p_t + h_t + \dots + h_n$ and the constant $K = \sum_{t=1}^n h_t (\sum_{i=1}^t d_i)$.

Discrete Alternatives or Disjunctions

Suppose $x \in R^n$ satisfies $0 \leq x \leq u$, and either $a^1 x \leq b_1$ or $a^2 x \leq b_2$ (see Figure 1.4). We introduce binary variables y_i for $i = 1, 2$. Then if $M \geq \max\{a^i x - b_i : 0 \leq x \leq u\}$ for $i = 1, 2$, we take as constraints:

$$\begin{aligned} a^i x - b_i \leq M(1 - y_i) \text{ for } i = 1, 2 \\ y_1 + y_2 = 1, y_i \in \{0, 1\} \text{ for } i = 1, 2 \\ 0 \leq x \leq u. \end{aligned}$$

Now if $y_1 = 1$, x satisfies $a^1 x \leq b_1$ whereas $a^2 x \leq b_2$ is inactive, and conversely if $y_2 = 1$.

Such disjunctions arise naturally in scheduling problems. Suppose that two jobs must be processed on the same machine and cannot be processed

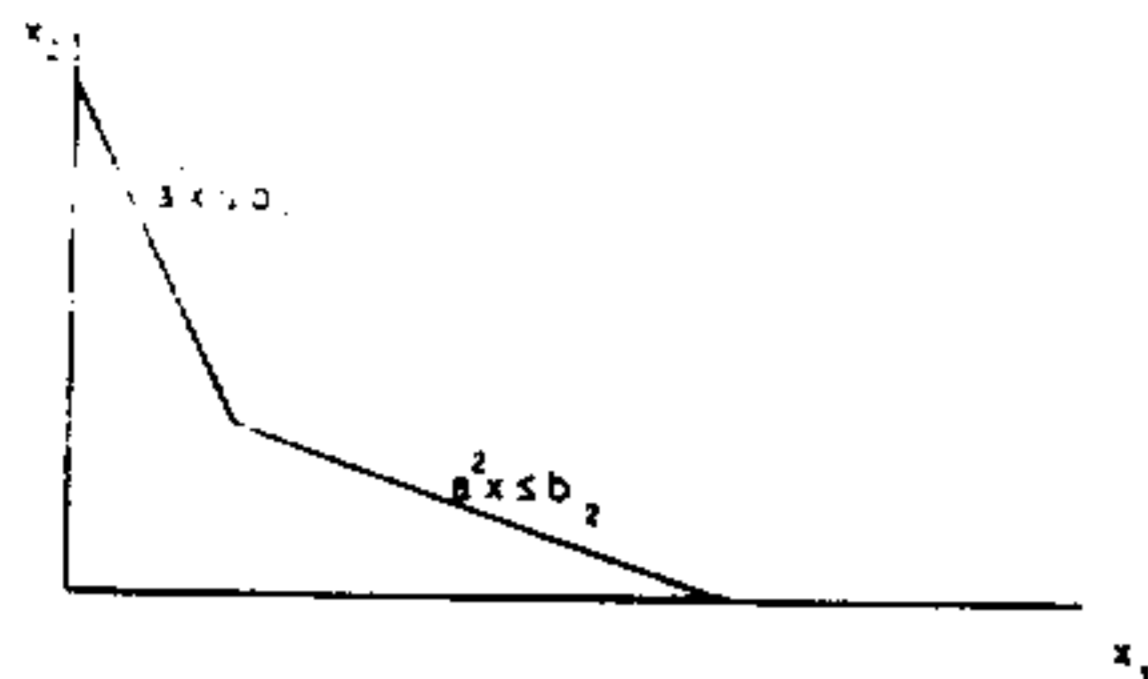


Fig. 1.4 Either/or constraints

simultaneously. If p_i are the processing times, and the variables t_i the start times for $i = 1, 2$, then either job 1 precedes job 2 and so $t_2 \geq t_1 + p_1$, or job 2 comes first and $t_1 \geq t_2 + p_2$.

1.6 ALTERNATIVE FORMULATIONS

In the two previous sections we have formulated a small number of integer programs for which it is not too difficult to verify that the formulations are correct. Here and in the next section we examine alternative formulations and try to understand why some might be better than others. First we make precise what we mean by a formulation.

Definition 1.1 A subset of R^n described by a finite set of linear constraints $P = \{x \in R^n : Ax \leq b\}$ is a *polyhedron*.

Definition 1.2 A polyhedron $P \subseteq R^{n+p}$ is a *formulation* for a set $X \subseteq Z^n \times R^p$ if and only if $X = P \cap (Z^n \times R^p)$.

Note that this definition is restrictive. For example we saw in modeling fixed costs in the previous section that we did not model the set $X = \{(0, 0), (x, 1)\}$ for $0 < x \leq C$, but the set $X \cup \{(0, 1)\}$.

Example 1.2 In Figure 1.5 we show two different formulations for the set:

$$X = \{(1, 1), (2, 1), (3, 1), (1, 2), (2, 2), (3, 2), (2, 3)\}. \quad \blacksquare$$

Note that it would not be easy to write a formulation for the set $X \setminus \{(2, 2)\}$ without introducing many more variables, see Exercise 1.2.

Equivalent Formulations for a 0-1 Knapsack Set

Consider the set of points $X =$

$$\{(0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1)\}.$$

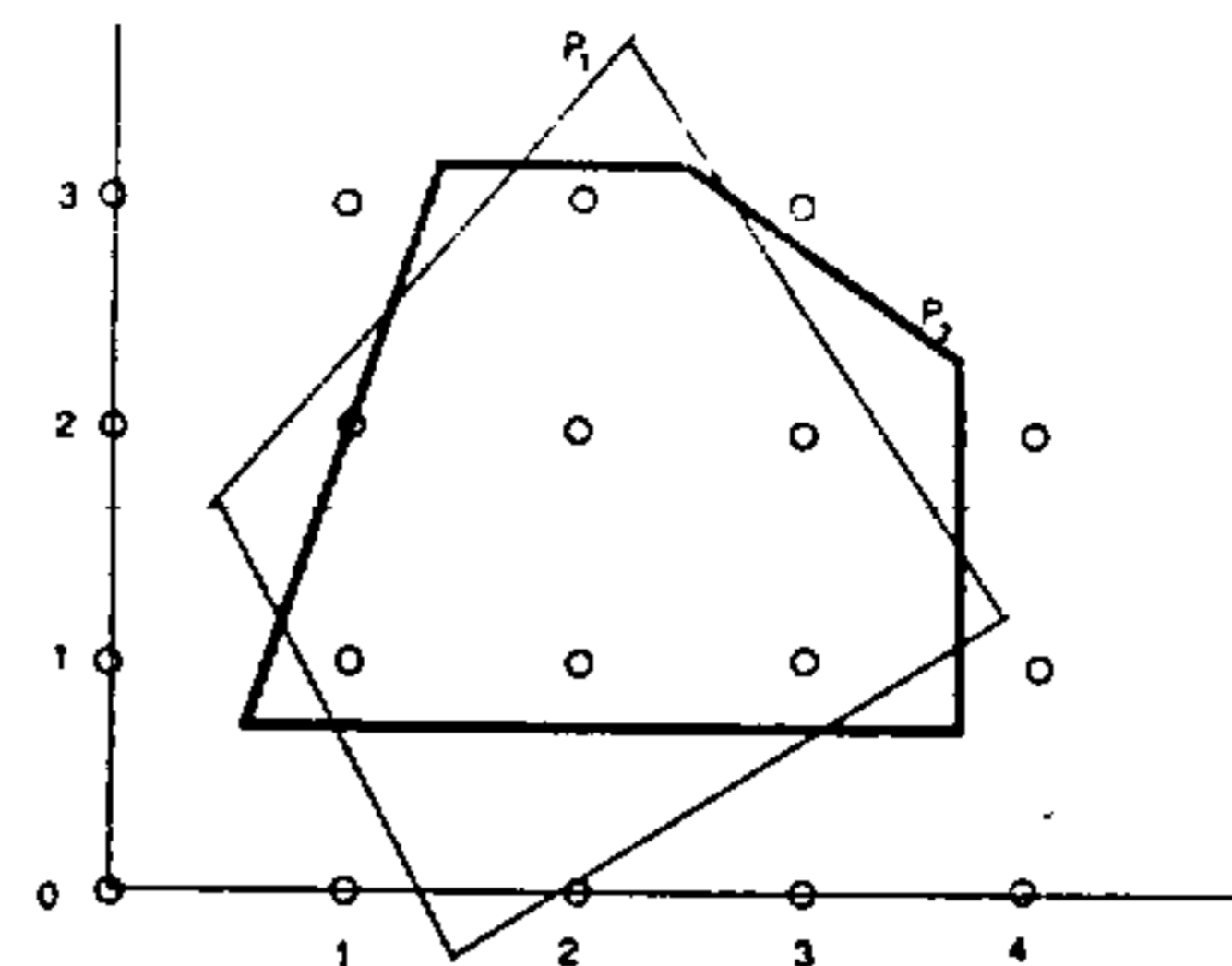


Fig. 1.5 Two different formulations of an IP

Check that the three polyhedra below are formulations for X .

$$P_1 = \{x \in R^4 : 0 \leq x \leq 1, 83x_1 + 61x_2 + 49x_3 + 20x_4 \leq 100\},$$

$$P_2 = \{x \in R^4 : 0 \leq x \leq 1, 4x_1 + 3x_2 + 2x_3 + 1x_4 \leq 4\},$$

$$P_3 = \{x \in R^4 : \begin{array}{cccc} 4x_1 & +3x_2 & +2x_3 & +1x_4 & \leq 4 \\ 1x_1 & +1x_2 & +1x_3 & & \leq 1 \\ 1x_1 & & & +1x_4 & \leq 1 \\ & & & & 0 \leq x \leq 1 \end{array}\}.$$

An Equivalent Formulation for UFL

For fixed j , consider the constraints:

$$\sum_{i \in M} x_{ij} \leq my_j, y_j \in \{0, 1\}, 0 \leq x_{ij} \leq 1 \text{ for } i \in M.$$

Logically these constraints express the condition: if any $x_{ij} > 0$, then $y_j = 1$, or stated a little differently: for each i , if $x_{ij} > 0$, then $y_j = 1$. This immediately suggests a different set of constraints:

$$0 \leq x_{ij} \leq y_j \text{ for } i \in M, y_j \in \{0, 1\}.$$

This leads to the alternative formulation:

$$\min \sum_{i=1}^m \sum_{j=1}^n c_{ij}x_{ij} + \sum_{j=1}^n f_j y_j \tag{1.1}$$

$$\sum_{j=1}^n x_{ij} = 1 \text{ for } i \in M \tag{1.2}$$

$$x_{ij} \leq y_j \text{ for } i \in M, j \in N \tag{1.3}$$

$$x_{ij} \geq 0 \text{ for } i \in M, j \in N, y_j \in \{0, 1\} \text{ for } j \in N. \tag{1.4}$$

In the two examples we have just examined, the variables have been the same in each formulation, and we have essentially modified or added constraints. Another possible approach is to add or choose different variables, in which case we talk of *extended formulations*.

An Extended Formulation for ULS

What variables, other than production and stock levels, might be useful in describing an optimal solution of the lot-sizing problem? Suppose we would like to know when the items being produced now will actually be used to satisfy demand. Following the same steps as before, this leads to:

Definition of the variables.

w_{it} is the amount produced in period i to satisfy demand in period t
 $y_t = 1$ if production occurs in period t (as above), and $y_t = 0$ otherwise.

Definition of the constraints.

Satisfaction of demand in period t :

$$\sum_{i=1}^t w_{it} = d_t \text{ for all } t.$$

Variable upper-bound constraints:

$$w_{it} \leq d_t y_i \text{ for all } i, t, i \leq t.$$

The variables are mixed:

$$w_{it} \geq 0 \text{ for all } i, t, i \leq t, y_t \in \{0, 1\} \text{ for all } t.$$

Definition of the objective function.

$$\min \sum_{i=1}^n \sum_{t=i}^n c_t w_{it} + \sum_{t=1}^n f_t y_t.$$

Note that here we are again using modified costs with c_t the production cost in period t and zero storage costs. It is optional whether we choose to define the old variables in terms of the new by adding defining equalities:

$$x_i = \sum_{t=i}^n w_{it}.$$

1.7 GOOD AND IDEAL FORMULATIONS

In Figure 1.5 we show two different formulations of the same problem. We have also seen two possible formulations for the uncapacitated facility location

problem. Geometrically we can see that there must be an infinite number of formulations, so how can we choose between them?

The geometry again helps us to find an answer. Look at Figure 1.6 in which we have repeated the two formulations shown in Figure 1.5, and added a third one P_3 . Formulation P_3 is *ideal*, because now if we solve a linear program over P_3 , the optimal solution is at an extreme point. In this ideal case each extreme point is integer and so the *IP* is solved.

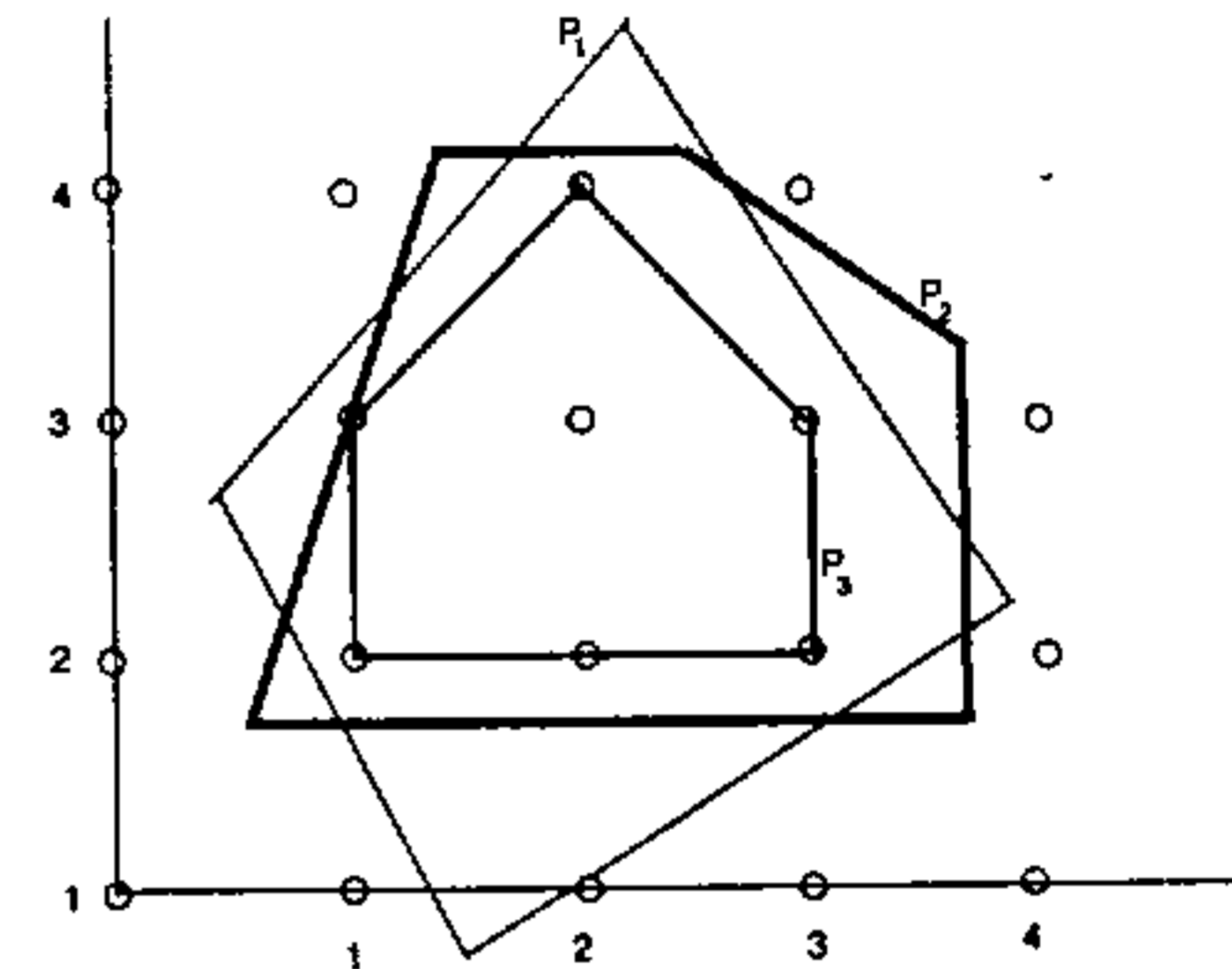


Fig. 1.6 The ideal formulation

We can now formalize this idea.

Definition 1.3 Given a set $X \subseteq R^n$, the *convex hull* of X , denoted $\text{conv}(X)$, is defined as: $\text{conv}(X) = \{x : x = \sum_{i=1}^t \lambda_i x^i, \sum_{i=1}^t \lambda_i = 1, \lambda_i \geq 0 \text{ for } i = 1, \dots, t \text{ over all finite subsets } \{x^1, \dots, x^t\} \text{ of } X\}$.

Proposition 1.1 $\text{conv}(X)$ is a polyhedron.

Proposition 1.2 The extreme points of $\text{conv}(X)$ all lie in X .

Because of these two results, we can replace the *IP*: $\{\max cx : x \in X\}$ by the equivalent linear program: $\{\max cx : x \in \text{conv}(X)\}$. This ideal reduction to a linear program also holds for unbounded integer sets $X = \{x : Ax \leq b, x \geq 0 \text{ and integer}\}$, and mixed integer sets $X = \{(x, y) : Ax + Gy \leq b, x \geq 0, y \geq 0 \text{ and integer}\}$ with A, G, b rational. However, whether X is bounded or not, this is in general only a theoretical solution, because in most cases there is such an enormous (exponential) number of inequalities needed to describe $\text{conv}(X)$, and there is no simple characterization for them.

So we might rather ask the question: Given two formulations P_1 and P_2 for X , when can we say that one is *better* than the other? Because the ideal solution $\text{conv}(X)$ has the property that $X \subseteq \text{conv}(X) \subseteq P$ for all formulations P , this suggests the following definition.

Definition 1.4 Given a set $X \subseteq R^n$, and two formulations P_1 and P_2 for X , P_1 is a *better formulation* than P_2 if $P_1 \subset P_2$.

In the next chapter we will see that this turns out to be a useful definition, and a very important idea for the development of effective formulations. For the moment we examine some of our formulations in the light of these definitions.

Equivalent Formulations for a Knapsack Set

Looking again at the set $X =$

$\{(0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1)\}$
and the three formulations,

$$P_1 = \{x \in R^4 : 0 \leq x \leq 1, 83x_1 + 61x_2 + 49x_3 + 20x_4 \leq 100\},$$

$$P_2 = \{x \in R^4 : 0 \leq x \leq 1, 4x_1 + 3x_2 + 2x_3 + 1x_4 \leq 4\},$$

$$P_3 = \{x \in R^4 : \begin{array}{cccc} 4x_1 & +3x_2 & +2x_3 & +1x_4 & \leq 4 \\ 1x_1 & +1x_2 & +1x_3 & & \leq 1 \\ 1x_1 & & & +1x_4 & \leq 1 \\ & & & 0 \leq x & \leq 1 \end{array}\},$$

it is easily seen that $P_3 \subset P_2 \subset P_1$, and it can be checked that $P_3 = \text{conv}(X)$, and P_3 is thus an ideal formulation.

Formulations for UFL

Let P_1 be the formulation given in Section 1.4 with a single constraint

$$\sum_{i \in M} x_{ij} \leq my_j$$

for each j , and P_2 be the formulation given in Section 1.5 with the m constraints

$$x_{ij} \leq y_j \text{ for } i \in M$$

for each j . Note that if a point (x, y) satisfies the constraints $x_{ij} \leq y_j$ for $i \in M$, then summing over $i \in M$ shows that it also satisfies the constraints $\sum_i x_{ij} \leq my_j$. Thus $P_2 \subseteq P_1$. To show that $P_2 \subset P_1$, we need to find a point in P_1 that is not in P_2 . Suppose for simplicity that n divides m , that is, $m = kn$ with $k \geq 2$ and integer. Then a point in which each depot serves k clients, $x_{ij} = 1$ for $i = k(j-1) + 1, \dots, k(j-1) + k, j = 1, \dots, n, x_{ij} = 0$ otherwise, and $y_j = k/m$ for $j = 1, \dots, n$ lies in $P_1 \setminus P_2$.

The two formulations that we have seen for the lot-sizing problem *ULS* have different variables, so it is not immediately obvious how they can be compared. We now examine this question briefly.

Formally, assuming for simplicity that all the variables are integer, we have a first formulation

$$\min\{cx : x \in P \cap Z^n\}$$

with $P \subseteq R^n$, and a second extended formulation

$$\min\{cx : (x, w) \in Q \cap (Z^n \times R^p)\}$$

with $Q \subseteq R^n \times R^p$.

Definition 1.5 Given a polyhedron $Q \subseteq (R^n \times R^p)$, the *projection* of Q onto the subspace R^n , denoted $\text{proj}_x Q$, is defined as:

$$\text{proj}_x Q = \{x \in R^n : (x, w) \in Q \text{ for some } w \in R^p\}.$$

Thus $\text{proj}_x Q \subseteq R^n$ is the formulation obtained from Q in the space R^n of the original x variables, and it allows us to compare Q with other formulations $P \subseteq R^n$.

Comparing Formulations for ULS

We can now compare the formulation P_1 :

$$\begin{aligned} s_{t-1} + x_t &= d_t + s_t \text{ for } t = 1, \dots, n \\ x_t &\leq My_t \text{ for } t = 1, \dots, n \\ s_0 = 0, s_t, x_t &\geq 0, 0 \leq y_t \leq 1 \text{ for all } t, \end{aligned}$$

and $P_2 = \text{proj}_{x,s,y} Q_2$, where Q_2 takes the form:

$$\begin{aligned} \sum_{i=1}^t w_{it} &= d_t \text{ for all } t \\ w_{it} &\leq d_t y_i \text{ for all } i, t, i \leq t \\ x_i &= \sum_{t=i}^n w_{it} \text{ for all } i \\ w_{it} &\geq 0 \text{ for all } i, t \text{ with } i \leq t \\ 0 &\leq y_t \leq 1 \text{ for all } t. \end{aligned}$$

It can be shown that P_2 describes the convex hull of solutions to the problem, and is thus an ideal formulation. It is easily seen that $P_2 \subset P_1$. For instance the point $x_t = d_t, y_t = d_t/M$ for all t is an extreme point of P_1 that is not in P_2 .

1.8 NOTES

Introduction

Much of the material in this book except for the last three or four chapters is treated in greater detail and at a more advanced level in [NemWol88]. An even more advanced theoretical treatment of integer programming is given in [Sch86]. Other recent books include [ParRar88], [SalMat89] and [Sie96]. For the related topic of combinatorial optimization, an undergraduate text is [PapSti82], and the recent graduate-level book [CooCunetal97] is recommended.

An important source of references and surveys are the annotated bibliographies of [OheLenRin85] and [DeLAMafMar97]. The journals publishing articles on integer programming cover a wide spectrum from theory to applications. The more theoretical include *Mathematical Programming*, *Mathematics of Operations Research*, the *SIAM Journal on Discrete Mathematics*, the *SIAM Journal on Optimization* and *Discrete Applied Mathematics*, while *Operations Research*, the *European Journal of Operations Research*, *Networks*, *Management Science*, and so on, contain more integer programming applications. For a survey of integer programming, see [NemWol89].

For those unfamiliar with linear programming, [Dan63] is the ultimate classic, and the book [Chv83] is exceptional. Recently many books treating interior point as well as simplex methods have appeared, including [RooTerVia97] and [Vdbei96]. For graph theory the books [Ber73] and [BonMur76] are classics. For network flows [ForFul62] and [Law76] remain remarkably stimulating, while [AhuMagOrl93] is very comprehensive and up-to-date.

In the notes at the end of each chapter, we will cite only selected references: a few of the important original sources, and also some recent surveys and articles that can be used for further reading.

Notes for the Chapter

The importance of formulations in integer programming has only become fully apparent in the last twenty years. The book [Wil78] is partly devoted to this topic, but is now a little dated. Certain special classes of problems that are often tackled as integer programs have had books dedicated to them: knapsack problems [MarTot90], traveling salesman problems [Lawetal85], location problems [MirFra90], network flows [AhuMagOrl93], network models [Balletal95a] and network routing [Balletal95b], and production and inventory [GraRinZip93]. For scheduling problems the recent survey [QueSch94] contains a variety of integer programming formulations.

1.9 EXERCISES

1. Suppose that you are interested in choosing a set of investments $\{1, \dots, 7\}$ using 0-1 variables. Model the following constraints:

- (i) You cannot invest in all of them.
- (ii) You must choose at least one of them.
- (iii) Investment 1 cannot be chosen if investment 3 is chosen.
- (iv) Investment 4 can be chosen only if investment 2 is also chosen.
- (v) You must choose either both investments 1 and 5 or neither.
- (vi) You must choose either at least one of the investments 1,2,3, or at least two investments from 2,4,5,6.

2. Formulate the following as mixed integer programs:

- (i) $u = \min\{x_1, x_2\}$, assuming that $0 \leq x_j \leq C$ for $j = 1, 2$
- (ii) $v = |x_1 - x_2|$ with $0 \leq x_j \leq C$, for $j = 1, 2$
- (iii) the set $X \setminus \{x^*\}$ where $X = \{x \in Z^n : Ax \leq b\}$ and $x^* \in X$.

3. Modeling disjunctions.

(i) Extend the formulation of discrete alternatives of Section 1.5 to the union of two polyhedra $P_k = \{x \in R^n : A^k x \leq b^k, 0 \leq x \leq u\}$ for $k = 1, 2$ where $\max_k \max_i \{a_i^k x - b_i^k : 0 \leq x \leq u\} \leq M$.

(ii) Show that an extended formulation for $P_1 \cup P_2$ is

$$\begin{aligned} x &= z^1 + z^2 \\ A^k z^k &\leq b^k y^k \text{ for } k = 1, 2 \\ 0 \leq z^k &\leq u y^k \text{ for } k = 1, 2 \\ y^1 + y^2 &= 1 \\ z^k &\in R^n, y^k \in B^1 \text{ for } k = 1, 2. \end{aligned}$$

4. Show that

$$\begin{aligned} X &= \{x \in B^4 : 97x_1 + 32x_2 + 25x_3 + 20x_4 \leq 139\} \\ &= \{x \in B^4 : 2x_1 + x_2 + x_3 + x_4 \leq 3\} \\ &= \{x \in B^4 : x_1 + x_2 + x_3 \leq 2 \\ &\quad x_1 + x_2 + x_4 \leq 2 \\ &\quad x_1 + x_3 + x_4 \leq 2\}. \end{aligned}$$

5. John Dupont is attending a summer school where he must take four courses per day. Each course lasts an hour, but because of the large number of students, each course is repeated several times per day by different teachers. Section i of course k denoted (i, k) meets at the hour t_{ik} , where courses start on the hour between 10 a.m. and 7 p.m. John's preferences for when he takes courses are influenced by the reputation of the teacher, and also the time of

day. Let p_{ik} be his preference for section (i, k) . Unfortunately, due to conflicts, John cannot always choose the sections he prefers.

- (i) Formulate an integer program to choose a feasible course schedule that maximizes the sum of John's preferences.
- (ii) Modify the formulation, so that John never has more than two consecutive hours of classes without a break.
- (iii) Modify the formulation, so that John chooses a schedule in which he starts his day as late as possible.

6. Prove that the set of feasible solutions to the formulation of the traveling salesman problem in Section 1.2 is precisely the set of incidence vectors of tours.

7. The QED Company must draw up a production program for the next nine weeks. Jobs last several weeks and once started must be carried out without interruption. During each week a certain number of skilled workers are required to work full-time on the job. Thus if job i lasts p_i weeks, $l_{i,u}$ workers are required in week u for $u = 1, \dots, p_i$. The total number of workers available in week t is L_t . Typical job data $(i, p_i, l_{i1}, \dots, l_{ip_i})$ is shown below.

Job	Length	Week1	Week2	Week3	Week4
1	3	2	3	1	-
2	2	4	5	-	-
3	4	2	4	1	5
4	4	3	4	2	2
5	3	9	2	3	-

- (i) Formulate the problem of finding a feasible schedule as an *IP*.
- (ii) Formulate when the objective is to minimize the maximum number of workers used during any of the nine weeks.
- (iii) Job 1 must start at least two weeks before job 3. Formulate.
- (iv) Job 4 must start not later than one week after job 5. Formulate.
- (v) Jobs 1 and 2 both need the same machine, and cannot be carried out simultaneously. Formulate.

8. Show that the uncapacitated facility location problem of Section 1.5 with a set $N = \{1, \dots, n\}$ of depots can be written as the *COP*

$$\min_{S \subseteq N} \{c(S) + \sum_{j \in S} f_j\}$$

where $c(S) = \sum_{i=1}^m \min_{j \in S} c_{ij}$.

9. Show that the covering problem of Section 1.3 can be written as the *COP*

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} f_j : v(S) = v(N) \right\}$$

where $v(S) = \sum_{i=1}^m \min\{\sum_{j \in S} a_{ij}, 1\}$.

10. A set of n jobs must be carried out on a single machine that can do only one job at a time. Each job j takes p_j hours to complete. Given job weights w_j for $j = 1, \dots, n$, in what order should the jobs be carried out so as to minimize the weighted sum of their start times? Formulate this scheduling problem as a mixed integer program.

11. Using a mixed integer programming system, solve an instance of the uncapacitated facility location problem, where f_j is the cost of opening depot j , and c_{ij} is the cost of satisfying all client i 's demand from depot j , with $f = (4, 3, 4, 4, 7)$ and

$$(c_{ij}) = \begin{pmatrix} 12 & 13 & 6 & 0 & 1 \\ 8 & 4 & 9 & 1 & 2 \\ 2 & 6 & 6 & 0 & 1 \\ 3 & 5 & 2 & 1 & 8 \\ 8 & 0 & 5 & 10 & 8 \\ 2 & 0 & 3 & 4 & 1 \end{pmatrix}$$

12. The symmetric traveling salesman problem is a *TSP* in which $c_{ij} = c_{ji}$ for all $(i, j) \in A$. Consider an instance on the graph shown in Figure 1.7, where the missing edges have a very high cost. Solve with a mixed integer programming system.

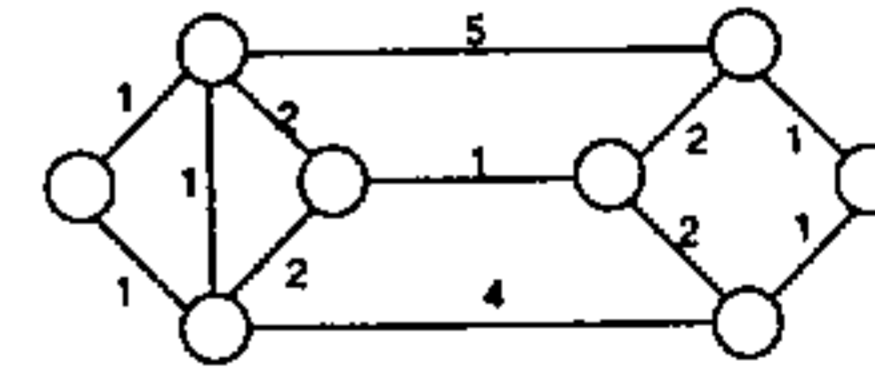


Fig. 1.7 TSP instance

13. Formulate and solve an instance of the lot-sizing problem over 6 periods, with demands $(6, 7, 4, 6, 3, 8)$, unit production costs $(3, 4, 3, 4, 4, 5)$, unit storage costs $(1, 1, 1, 1, 1, 1)$, set-up costs $(12, 15, 30, 23, 19, 45)$, and a maximum production capacity of 10 items per period.

14. Formulate and solve the problem of placing N queens on an N by N chessboard such that no two queens share any row, column, or diagonal.

15. (Projection). Let $Q = \{(x, y) \in R_+^n \times R_+^p : Ax + Gy \leq b\}$. Use Farkas' Lemma to show that

$$\text{proj}_y(Q) = \{y \in R_+^p : v^t(b - Gy) \geq 0 \text{ for } t = 1, \dots, T\}$$

where $\{v^t\}_{t=1}^T$ are the extreme rays of $V = \{v \in R_+^m : vA \geq 0\}$.

Show that if $\text{proj}_y(Q) \neq \emptyset$,

$$\max\{cx + hy : (x, y) \in Q\} = \max\left\{\min_{y \in \text{proj}_y(Q)} \sum_{s=1}^S u^s(b - Gy) + hy : y \in \text{proj}_y(Q)\right\}$$

where $\{u^s\}_{s=1}^S$ are the extreme points of $U = \{u \in R_+^m : uA \geq c\}$.

2

Optimality, Relaxation, and Bounds

2.1 OPTIMALITY AND RELAXATION

Given an *IP* or *COP*

$$z = \max\{c(x) : x \in X \subseteq Z^n\},$$

how is it possible to prove that a given point x^* is optimal? Put differently, we are looking for some optimality conditions that will provide stopping criteria in an algorithm for *IP*.

The "naive" but nonetheless important reply is that we need to find a lower bound $\underline{z} \leq z$ and an upper bound $\bar{z} \geq z$ such that $\underline{z} = \bar{z} = z$. Practically, this means that any algorithm will find a decreasing sequence

$$\bar{z}_1 > \bar{z}_2 > \dots > \bar{z}_s \geq z$$

of upper bounds, and an increasing sequence

$$\underline{z}_1 < \underline{z}_2 < \dots < \underline{z}_t \leq z,$$

of lower bounds, and stop when

$$\bar{z}_s - \underline{z}_t \leq \epsilon,$$

where ϵ is some suitably chosen small nonnegative value (see Figure 2.1). Thus we need to find ways of deriving such upper and lower bounds.

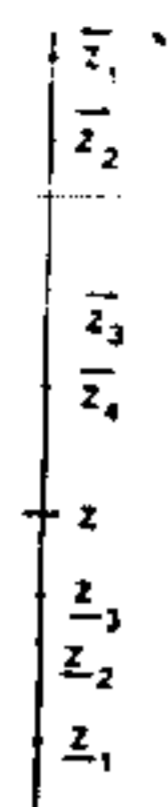


Fig. 2.1 Bounds for IP

Primal Bounds

Every feasible solution $x^* \in X$ provides a lower bound $z = c(x^*) \leq z$. This is essentially the only way we know to obtain lower bounds. For some IP problems, finding feasible solutions is easy, and the real question is how to find good solutions. For instance in the traveling salesman problem, if the salesman is allowed to travel between any pair of cities, any permutation of the cities leads to a feasible tour, and it suffices to evaluate the length of the tour to have a primal bound on z . Some simple ways to find feasible solutions and then improve them are discussed later in this chapter. For other IPs, finding feasible solutions may be very difficult (as difficult as the IP itself). This topic is raised again when we discuss complexity in Chapter 6, and heuristics to find primal bounds are treated in more detail in Chapter 12.

Dual Bounds

Finding upper bounds for a maximization problem (or lower bounds for a minimization problem) presents a different challenge. These are called *dual bounds* in contrast to the primal bounds for reasons that should become obvious in Section 2.5. The most important approach is by "relaxation," the idea being to replace a "difficult" max(min) IP by a simpler optimization problem whose optimal value is at least as large (small) as z .

For the "relaxed" problem to have this property, there are two obvious possibilities:

- (i) Enlarge the set of feasible solutions so that one optimizes over a larger set, or
- (ii) Replace the max(min) objective function by a function that has the same or a larger (smaller) value everywhere.

Definition 2.1 A problem (RP) $z^R = \max\{f(x) : x \in T \subseteq R^n\}$ is a *relaxation* of (IP) $z = \max\{c(x) : x \in X \subseteq R^n\}$ if:

- (i) $X \subseteq T$, and
- (ii) $f(x) \geq c(x)$ for all $x \in X$.

Proposition 2.1 If RP is a relaxation of IP, $z^R \geq z$.

Proof. If x^* is an optimal solution of IP, $x^* \in X \subseteq T$ and $z = c(x^*) \leq f(x^*)$. As $x^* \in T$, $f(x^*)$ is a lower bound on z^R , and so $z \leq f(x^*) \leq z^R$. ■

The question then arises of how to construct interesting relaxations. One of the most useful and natural ones is the linear programming relaxation.

2.2 LINEAR PROGRAMMING RELAXATIONS

Definition 2.2 For the integer program $\max\{cx : x \in P \cap Z^n\}$ with formulation $P = \{x \in R_+^n : Ax \leq b\}$, the *linear programming relaxation* is the linear program $z^{LP} = \max\{cx : x \in P\}$.

As $P \cap Z^n \subseteq P$ and the objective function is unchanged, this is clearly a relaxation.

Example 2.1 Consider the integer program

$$\begin{aligned} z = \max \quad & 4x_1 - x_2 \\ & 7x_1 - 2x_2 \leq 14 \\ & \quad \quad x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x \in Z_+^2. \end{aligned}$$

To obtain a primal (lower) bound, observe that (2, 1) is a feasible solution, so we have the lower bound $z \geq 7$. To obtain a dual (upper) bound, consider the linear programming relaxation. The optimal solution is $x^* = (\frac{20}{7}, 3)$ with value $z^{LP} = \frac{59}{7}$. Thus we obtain an upper bound $z \leq \frac{59}{7}$. Observing that the optimal value must be integer, we can round down to the nearest integer and so obtain $z \leq 8$. ■

Note that the definition of better formulations is intimately related to that of linear programming relaxations. In particular better formulations give tighter (dual) bounds.

Proposition 2.2 Suppose P_1, P_2 are two formulations for the integer program $\max\{cx : x \in X \subseteq Z^n\}$ with P_1 a better formulation than P_2 , i.e. $P_1 \subset P_2$. If $z_i^{LP} = \max\{cx : x \in P_i\}$ for $i = 1, 2$ are the values of the associated linear programming relaxations, then $z_1^{LP} \leq z_2^{LP}$ for all c .

Relaxations do not just give dual bounds. They sometimes allow us to prove optimality.

Proposition 2.3 (i) If a relaxation RP is infeasible, the original problem IP is infeasible.

(ii) Let x^* be an optimal solution of RP . If $x^* \in X$ and $f(x^*) = c(x^*)$, then x^* is an optimal solution of IP .

Proof. (i) As RP is infeasible, $T = \phi$ and thus $X = \phi$.

(ii) As $x^* \in X$, $z \geq c(x^*) = f(x^*) = z^R$. As $z \leq z^R$, we obtain $c(x^*) = z = z^R$. ■

Example 2.2 The linear programming relaxation of the integer program:

$$\begin{aligned} \max & 7x_1 + 4x_2 + 5x_3 + 2x_4 \\ & 3x_1 + 3x_2 + 4x_3 + 2x_4 \leq 6 \\ & x \in B^4 \end{aligned}$$

has optimal solution $x^* = (1, 1, 0, 0)$. As x^* is integral, it solves the integer program. ■

2.3 COMBINATORIAL RELAXATIONS

Whenever the relaxed problem is a combinatorial optimization problem, we speak of a *combinatorial relaxation*. In many cases, such as (i)–(iii) below, the relaxation is an easy problem that can be solved rapidly. Some of the problems arising in this way are studied in Chapters 3 and 4. Here we illustrate with four examples.

(i) **The Traveling Salesman Problem.** We saw in formulating the traveling salesman problem with digraph $D = (V, A)$ and arc weights c_{ij} for $(i, j) \in A$ that the (Hamiltonian or salesman) tours are precisely the assignments (or permutations) containing no subtours. Thus

$$\begin{aligned} z^{TSP} &= \min_{T \subseteq A} \left\{ \sum_{(i,j) \in T} c_{ij} : T \text{ forms a tour} \right\} \geq \\ z^{ASS} &= \min_{T \subseteq A} \left\{ \sum_{(i,j) \in T} c_{ij} : T \text{ forms an assignment} \right\}. \end{aligned}$$

(ii) A closely related problem is the **Symmetric Traveling Salesman Problem (STSP)** specified by a graph $G = (V, E)$ and edge weights c_e for $e \in E$. The problem is to find an undirected tour of minimum weight. An interesting relaxation of this problem is obtained by observing that

a) Every tour consists of two edges adjacent to node 1, and a path through nodes $\{2, \dots, n\}$.

b) A path is a special case of a tree.

Definition 2.3 A *1-tree* is a subgraph consisting of two edges adjacent to node 1, plus the edges of a tree on nodes $\{2, \dots, n\}$.

Clearly every tour is a 1-tree, and thus

$$z^{STSP} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a tour} \right\} \geq$$

$$z^{1\text{-tree}} = \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a 1-tree} \right\}.$$

(iii) **The Quadratic 0–1 Problem** is the problem:

$$\max \left\{ \sum_{i,j:1 \leq i < j \leq n} q_{ij} x_i x_j - \sum_{j=1}^n p_j x_j, x \neq 0, x \in B^n \right\}.$$

Replacing all terms $q_{ij} x_i x_j$ with $q_{ij} < 0$ by 0 gives a relaxation

$$z^R = \max \left\{ \sum_{i,j:1 \leq i < j \leq n} \max\{q_{ij}, 0\} x_i x_j - \sum_{j=1}^n p_j x_j, x \neq 0, x \in B^n \right\}.$$

In Chapter 9, it will be shown how this relaxation can be solved as a series of maximum flow problems.

(iv). **The Knapsack Problem.** A relaxation of the set $X = \{x \in Z_+^n : \sum_{j=1}^n a_j x_j \leq b\}$ is the set

$$X' = \{x \in Z_+^n : \sum_{j=1}^n [a_j] x_j \leq [b]\}$$

where $[a]$ is the largest integer less than or equal to a .

2.4 LAGRANGIAN RELAXATION

Suppose we are given an integer program (IP) in the form $z = \max\{cx : Ax \leq b, x \in X \subseteq Z^n\}$. If the problem is too difficult to solve directly, one possibility is just to drop the constraints $Ax \leq b$. Clearly the resulting problem: $z' = \max\{cx : x \in X\}$ is a relaxation of IP . In the asymmetric traveling salesman problem above, the assignment problem is obtained by dropping the subtour constraints. An important extension of this idea, dealt with in much greater detail in Chapter 10, is not just to drop complicating constraints, but then to add them into the objective function with Lagrange multipliers (dual variables).

Proposition 2.4 Let $z(u) = \max\{cx + u(b - Ax) : x \in X\}$. Then $z(u) \geq z$ for all $u \geq 0$.

Proof. Let x^* be optimal for IP . As x^* is feasible in IP , $x^* \in X$. Again by feasibility $Ax^* \leq b$, and thus as $u \geq 0$, $cx^* \leq cx^* + u(b - Ax^*) \leq z(u)$ where the last inequality is by definition of $z(u)$. ■

2.5 DUALITY

For linear programs duality provides a standard way to obtain upper bounds. It is therefore natural to ask whether it is possible to find duals for integer programs. The important property of a dual is that the value of any feasible solution provides an upper bound on the objective value z . This suggests the following definition.

Definition 2.4 The two problems

$$(IP) \quad z = \max\{c(x) : x \in X\}$$

$$(D) \quad w = \min\{\omega(u) : u \in U\}$$

form a (weak)-dual pair if $c(x) \leq \omega(u)$ for all $x \in X$ and all $u \in U$. When $z = w$, they form a strong-dual pair.

The advantage of a dual problem as opposed to a relaxation is that any dual feasible solution provides an upper bound on z , whereas a relaxation of IP must be solved to optimality to provide such a bound. Do such dual problems exist?

Not surprisingly, a linear programming relaxation immediately leads to a weak dual.

Proposition 2.5 The integer program $z = \max\{cx : Ax \leq b, x \in Z_+^n\}$ and the linear program $w^{LP} = \min\{ub : uA \geq c, u \in R_+^m\}$ form a weak dual pair.

By analogy with Proposition 2.3, dual problems sometimes allow us to prove optimality.

Proposition 2.6 Suppose that IP and D are a weak-dual pair.

- (i) If D is unbounded, IP is infeasible.
- (ii) If $x^* \in X$ and $u^* \in U$ satisfy $c(x^*) = \omega(u^*)$, then x^* is optimal for IP and u^* is optimal for D .

We now present another example of a dual pair of problems.

A Matching Dual

Given a graph $G = (V, E)$, a matching $M \subseteq E$ is a set of disjoint edges. A covering by nodes is a set $R \subseteq V$ of nodes such that every edge has at least one endpoint in R .

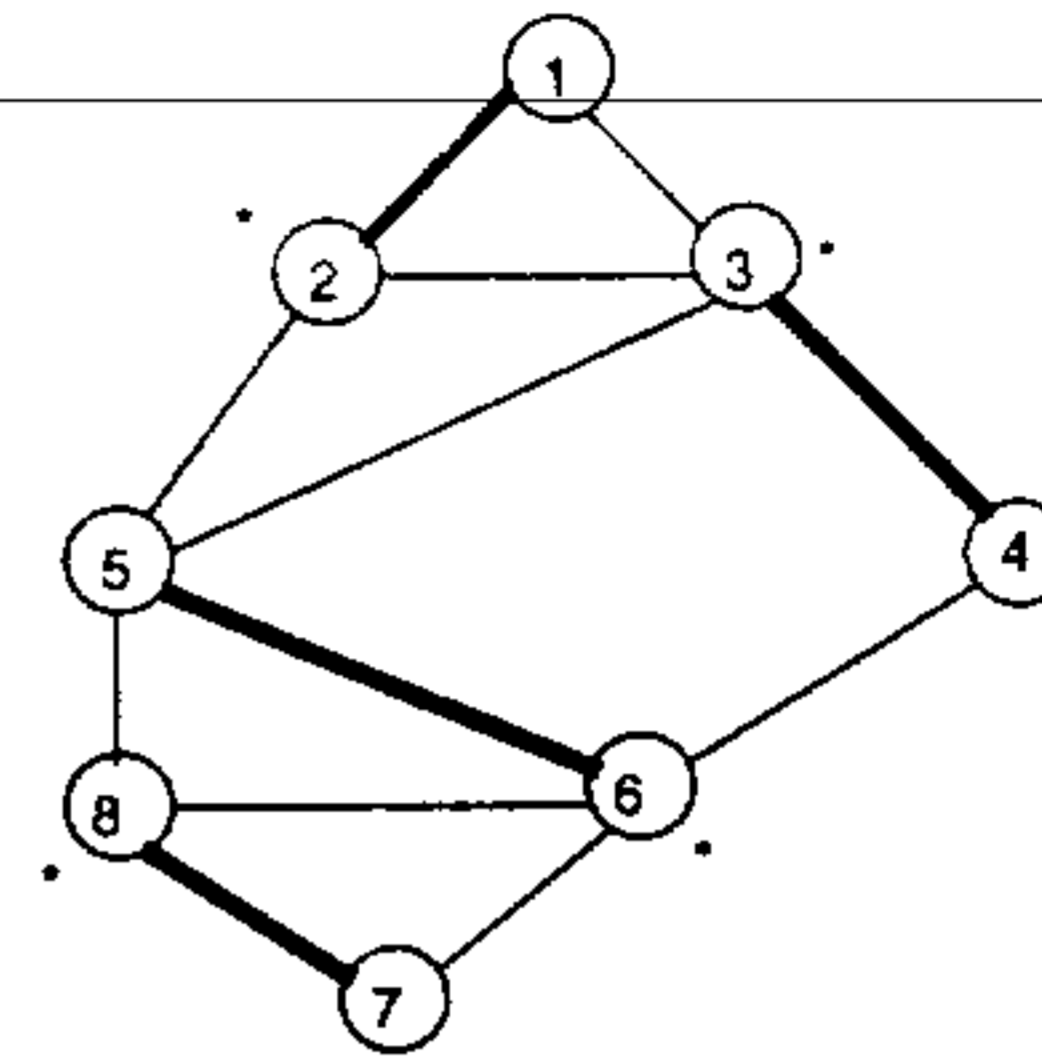


Fig. 2.2 Matching and cover by nodes

A graph on eight nodes is shown in Figure 2.2. The edges $(1, 2)$, $(3, 4)$, $(5, 6)$, and $(7, 8)$ form a matching and the nodes $\{2, 3, 6, 8\}$ a cover by nodes.

Proposition 2.7 The problem of finding a maximum cardinality matching:

$$\max_{M \subseteq E} \{|M| : M \text{ is a matching}\}$$

and the problem of finding a minimum cardinality covering by nodes:

$$\min_{R \subseteq V} \{|R| : R \text{ is a covering by nodes}\}$$

form a weak-dual pair.

Proof. If M is a matching with $M = \{(i_1, j_1), \dots, (i_k, j_k)\}$, then the $2k$ nodes $\{i_1, j_1, \dots, i_k, j_k\}$ are distinct, and any covering by nodes R must contain at least one node from each pair $\{i_s, j_s\}$ for $s = 1, \dots, k$. Therefore $|R| \geq k = |M|$. ■

We can also establish this result using linear programming duality.

Definition 2.5 The node-edge incidence matrix of a graph $G = (V, E)$ is an $n = |V|$ by $m = |E|$ 0-1 matrix A with $a_{j,e} = 1$ if node j is an endpoint of edge e , and $a_{j,e} = 0$ otherwise.

The maximum cardinality matching problem can now be formulated as the integer program:

$$z = \max\{1x : Ax \leq 1, x \in Z_+^m\}$$

and the minimum cardinality covering by nodes problem as:

$$w = \min\{1y : yA \geq 1, y \in Z_+^n\}.$$

Let z^{LP} and w^{LP} be the values of their corresponding linear programming relaxations. Then $z \leq z^{LP} = w^{LP} \leq w$, and the duality is again established.

Example 2.3 It is easily seen that there is not a strong duality between the two problems. Consider the graph shown in Figure 2.3.

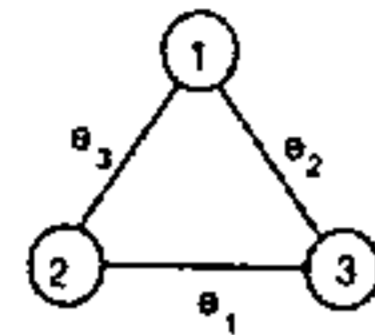


Fig. 2.3 Weak duality for matching

First observe that $z = 1$ and $w = 2$. What is more, $x_{e_1} = x_{e_2} = x_{e_3} = 1/2$ is feasible for the first LP relaxation, and $y_1 = y_2 = y_3 = 1/2$ is feasible for the second LP relaxation, so $z^{LP} = w^{LP} = 3/2$.

Later we will see that strong duality holds for this pair of problems when the graph G is bipartite.

2.6 PRIMAL BOUNDS: GREEDY AND LOCAL SEARCH

Now we briefly consider how to obtain primal bounds. The idea of a *greedy heuristic* is to construct a solution from scratch (the empty set), choosing at each step the item bringing the "best" immediate reward. We present two examples.

Example 2.4 The 0-1 Knapsack Problem. Consider the instance:

$$\begin{aligned} \max & 12x_1 + 8x_2 + 17x_3 + 11x_4 + 6x_5 + 2x_6 + 2x_7 \\ & 4x_1 + 3x_2 + 7x_3 + 5x_4 + 3x_5 + 2x_6 + 3x_7 \leq 9 \\ & x \in B^7. \end{aligned}$$

Noting that the variables are ordered so that $\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}$ for $j = 1, \dots, n-1$, a greedy solution is:

- (i) As $\frac{c_1}{a_1}$ is maximum and there is enough space (9 units) available, fix $x_1 = 1$.
- (ii) In the remaining problem, as $\frac{c_2}{a_2}$ is maximum and there is enough space (5=9-4) units available, fix $x_2 = 1$.
- (iii) As each item 3,4,5 in that order requires more space than the 2 = 5 - 3 units available, set $x_3 = x_4 = x_5 = 0$.
- (iv) As $\frac{c_6}{a_6}$ is maximum in the remaining problem, and there is enough space (2 units) available, fix $x_6 = 1$.
- (v) Set $x_7 = 0$ as no further space is available.

Therefore the greedy solution is $x^G = (1, 1, 0, 0, 0, 1, 0)$ with value $z^G = cz^G = 22$.

Example 2.5 The Symmetric Traveling Salesman Problem. Consider an instance with distance matrix:

$$(c_e) = \begin{pmatrix} - & 9 & 2 & 8 & 12 & 11 \\ & - & 7 & 19 & 10 & 32 \\ & & - & 29 & 18 & 6 \\ & & & - & 24 & 3 \\ & & & & - & 19 \\ & & & & & - \end{pmatrix}$$

Greedy examines the edges in order of nondecreasing length.

The cheapest edge is $e_1 = (1, 3)$ with $c_{e_1} = 2$. Select the edge by setting $x_{e_1} = 1$.

The next cheapest edge remaining is $e_2 = (4, 6)$ with $c_{e_2} = 3$. As edges e_1 and e_2 can appear together in a tour, set $x_{e_2} = 1$.

Set $x_{e_3} = 1$ where $e_3 = (3, 6)$ and $c_{e_3} = 6$.

Set $x_{e_4} = 0$ where $e_4 = (2, 3)$ as node 3 already has degree 2, and all three edges $(1, 3), (3, 6), (2, 3)$ cannot be in a tour.

Set $x_{e_5} = 0$ where $e_5 = (1, 4)$ as edge $(1, 4)$ forms a subtour with the edges already chosen.

Set $x_{e_6} = 1$ where $e_6 = (1, 2)$ and $c_{e_6} = 9$.

Continue as above, choosing edges $(2, 5)$ with length 10 and $(4, 5)$ with length 24 to complete the tour.

The greedy tour is $(1, 3, 6, 4, 5, 2, 1)$ with length $\sum_e c_e x_e = 54$.

Once an initial feasible solution, called the *incumbent*, has been found, it is natural to try to improve the solution. The idea of a *local search heuristic* is to define a neighborhood of solutions close to the incumbent. Then the best solution in the neighborhood is found. If it is better than the incumbent, it replaces it, and the procedure is repeated. Otherwise the incumbent is "locally optimal" with respect to the neighborhood, and the heuristic terminates. Again we present two examples.

Example 2.6 Uncapacitated Facility Location. Consider an instance with $m = 6$ clients and $n = 4$ depots, and costs as shown below:

$$(c_{ij}) = \begin{pmatrix} 6 & 2 & 3 & 4 \\ 1 & 9 & 4 & 11 \\ 15 & 2 & 6 & 3 \\ 9 & 11 & 4 & 8 \\ 7 & 23 & 2 & 9 \\ 4 & 3 & 1 & 5 \end{pmatrix} \text{ and } f_j = (21, 16, 11, 24).$$

Note that if $N = \{1, 2, 3, 4\}$ denotes the set of depots, and $S \subseteq N$ the set of open depots, the associated cost is

$$c(S) = \sum_{i=1}^6 \min_{j \in S} c_{ij} + \sum_{j \in S} f_j.$$

Thus if $S^0 = \{1, 2\}$ is the initial incumbent, $c(S^0) = (2 + 1 + 2 + 9 + 7 + 3) + 21 + 16 = 61$.

Now we need to define a neighborhood of S . One possibility is to consider as neighbors all sets obtained from S by the addition or removal of a single element:

$$Q(S) = \{T \subseteq N : T = S \cup \{j\} \text{ for } j \notin S \text{ or } T = S \setminus \{i\} \text{ for } i \in S\}.$$

Thus $Q(S^0) = \{\{1\}, \{2\}, \{1, 2, 3\}, \{1, 2, 4\}\}$ with costs $c(1) = 63, c(2) = 66, c(123) = 60, c(124) = 84$.

The new incumbent is $S^1 = \{1, 2, 3\}$ with $c(S^1) = 60$, and $Q(S^1) = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3, 4\}\}$.

The new incumbent is $S^2 = \{2, 3\}$ with $c(S^2) = 42$, and $Q(S^2) = \{\{2\}, \{3\}, \{1, 2, 3\}, \{2, 3, 4\}\}$.

The new incumbent is $S^3 = \{3\}$ with $c(S^3) = 31$, and $Q(S^3) = \{\{1, 3\}, \{2, 3\}, \{3, 4\}, \phi\}$.

There is no improvement, so $S^3 = \{3\}$ is a locally optimal solution. ■

Example 2.7 The Graph Equipartition Problem. Given a graph $G = (V, E)$ and $n = |V|$, the problem is to find a subset of nodes $S \subseteq V$ with $|S| = \lfloor \frac{n}{2} \rfloor$, for which the number of edges in the cutset $\delta(S, V \setminus S)$ is minimized, where $\delta(S, V \setminus S) = \{(i, j) \in E : i \in S, j \in V \setminus S\}$.

Again we need to define a neighborhood. As the feasible sets S are all of the same size, one possibility is to consider as neighbors all sets obtained by replacing one element in S by one element not in S :

$$Q(S) = \{T \subset V : |T \setminus S| = |S \setminus T| = 1\}.$$

We consider an instance on 6 nodes with edges $\{(1, 4), (1, 6), (2, 3), (2, 5), (2, 6), (3, 4), (3, 5), (4, 6)\}$.

Starting with $S^0 = \{1, 2, 3\}$, $c(S^0) = |\delta(S^0, V \setminus S^0)| = 6$ as $\delta(S^0, V \setminus S^0) = \{(1, 4), (1, 6), (2, 5), (2, 6), (3, 4), (3, 5)\}$.

Here $Q(S^0) = \{(1, 2, 4), (1, 2, 5), (1, 2, 6), (1, 3, 4), (1, 3, 5), (1, 3, 6), (2, 3, 4), (2, 3, 5), (2, 3, 6)\}$ with $c(T) = 6, 5, 4, 4, 5, 6, 5, 2, 5$ respectively.

The new incumbent is $S^1 = \{2, 3, 5\}$ with $c(S^1) = 2$.

$Q(S^1)$ does not contain a better solution, and so S^1 is locally optimal. ■

Other ways to generate primal feasible solutions are investigated in Chapter 12.

2.7 NOTES

2.1 The concept of a relaxation, and the complementary idea of a restriction are formalized in [GeoMar72].

2.2 Linear programming relaxations have been present ever since combinatorial problems were first formulated as linear integer programs [DanFulJoh54], [Dan57].

2.3 The assignment relaxation for the traveling salesman problem was already used in [Litetal63], and the 1-tree relaxation was introduced in [HelKar70].

2.4 Chapter 10 is on Lagrangian relaxation. Other relaxations studied include group or modular relaxations [Gom65] and Ch. II.3 in [NemWol88], and surrogate relaxations [Glo68].

2.5 A strong duality for the general matching problem appears in the classic paper [Edm65b], which has had a major influence on the development of combinatorial optimization. Several of the most beautiful results in this field are strong duality theorems; see [CooCunetal97]. For integer programs a general superadditive duality theory was developed in the seventies based on the work of [Gom69] and [GomJoh72]. See Ch. II.1 in [NemWol88].

2.6 The greedy and local exchange heuristics are formalized in Chapter 12, and other heuristic approaches are presented.

2.8 EXERCISES

1. Find a maximum cardinality matching in the graph of Figure 2.4 by inspection. Give a proof that the solution found is optimal.

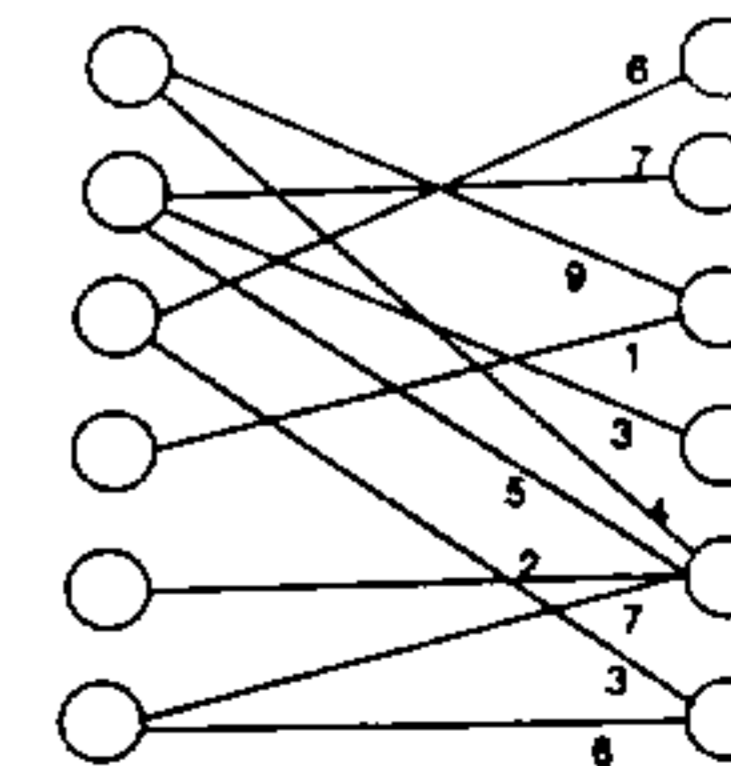


Fig. 2.4 Matching instance

2. A *stable set* is a set of nodes $U \subseteq V$ such that there are no edges between any two nodes of U . A *clique* is a set of nodes $U \subseteq V$ such that there is an edge between every pair of nodes in U . Show that the problem of finding a maximum cardinality stable set is dual to the problem of finding a minimum cover of the nodes by cliques. Use this observation to find bounds on the maximum size of a stable set in the graph shown in Figure 2.5.

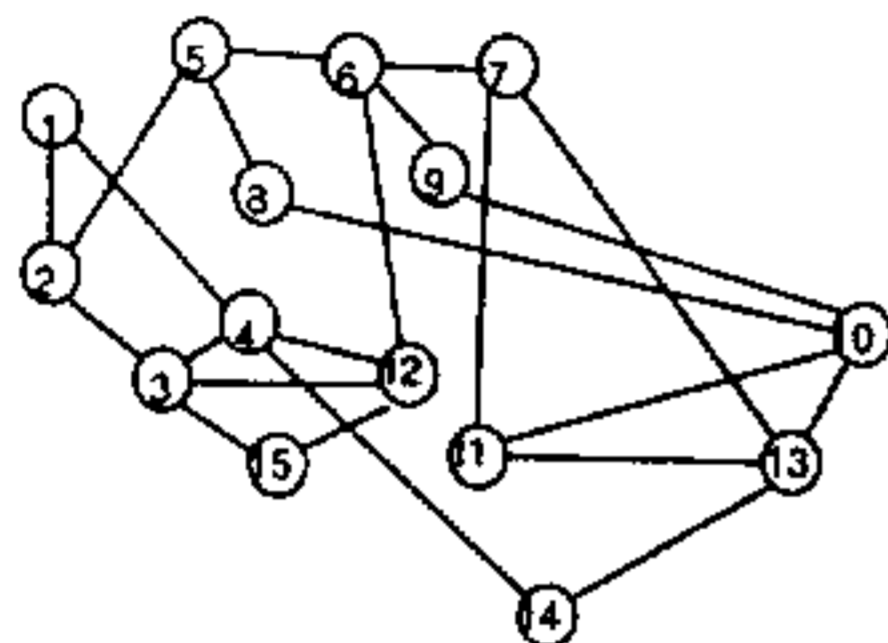


Fig. 2.5 Stable set instance

3. Find primal and dual bounds for the integer knapsack problem:

$$\begin{aligned} z &= \max 42x_1 + 26x_2 + 35x_3 + 71x_4 + 53x_5 \\ 14x_1 + 10x_2 + 12x_3 + 25x_4 + 20x_5 &\leq 69 \\ x &\in Z_+^5. \end{aligned}$$

4. Consider the 0-1 integer program:

$$(P_1) \quad \max\{cx : \sum_{j=1}^n a_{ij}x_j = b_i \text{ for } i = 1, \dots, m, x \in B^n\},$$

and the 0-1 equality knapsack problem

$$(P_2) \quad \max\{cx : \sum_{j=1}^n (\sum_{i=1}^m u_i a_{ij})x_j = \sum_{i=1}^m u_i b_i, x \in B^n\},$$

where $u \in R^m$. Show that P_2 is a relaxation of P_1 .

5. Consider the equality integer knapsack problem:

$$(P_1) \quad \min\{\sum_{j=1}^5 c_j x_j : \frac{7}{4}x_1 - \frac{2}{3}x_2 + \frac{5}{2}x_3 - \frac{5}{12}x_4 + \frac{19}{6}x_5 = \frac{8}{3}, x \in Z_+^5\}$$

(i) Show that the problem

$$(P_2) \quad \min\{\sum_{j=1}^5 c_j x_j : \frac{3}{4}x_1 + \frac{1}{3}x_2 + \frac{1}{2}x_3 + \frac{7}{12}x_4 + \frac{1}{6}x_5 = \frac{2}{3} + w, x \in Z_+^5, w \in Z_+^1\}$$

is a relaxation of P_1 .

(ii) Show that the problem

$$(P_3) \quad \min\{\sum_{j=1}^5 c_j x_j : \frac{3}{4}x_1 + \frac{1}{3}x_2 + \frac{1}{2}x_3 + \frac{7}{12}x_4 + \frac{1}{6}x_5 \geq \frac{2}{3}, x \in R_+^5\}$$

is a relaxation of P_2 .

6. Consider a directed graph $D = (V, A)$ with arc lengths $c_e \geq 0$ for $e \in A$. Taking two distinct nodes $s, t \in V$, consider the problem of finding a shortest path from s to t . Show that

$$\max\{\pi_t : \pi_j - \pi_i \leq c_{ij} \text{ for } e = (i, j) \in A, \pi \in R_+^{|V|}, \pi_s = 0\}$$

is a strong dual problem.

7. Apply a greedy heuristic to the instance of the uncapacitated facility location problem in Section 2.6.

8. Define greedy and local search heuristics for the maximum cardinality stable set problem discussed in Exercise 2.2.

9. Formulate the maximum cardinality matching problem of Figure 2.4 as an integer program and solve its linear programming relaxation. Find a maximum weight matching with the weights shown. Check that the linear programming relaxation is again integral.

3

Well-Solved Problems

3.1 PROPERTIES OF EASY PROBLEMS

Here we plan to study some integer and combinatorial optimization problems that are “well-solved” in the sense that an “efficient” algorithm is known for solving all instances of the problem. Clearly an instance with 1000 variables or data values ranging up to 10^{20} can be expected to take longer than an instance with 10 variables and integer data never exceeding 100. So we need to define what we mean by efficient.

For the moment we will be very imprecise and say that an algorithm on a graph $G = (V, E)$ with n nodes and m edges is *efficient* if, in the worst case, the algorithm requires $O(m^p)$ elementary calculations (such as additions, divisions, comparisons, etc) for some integer p , where we assume that $m \geq n$.

In considering the *COP* $\max\{cx : x \in X \subseteq R^n\}$, it is not just of interest to find a dual problem, but also to consider a related problem, called the separation problem.

Definition 3.1 The *Separation Problem* associated with *COP* is the problem: Given $x^* \in R^n$, is $x^* \in \text{conv}(X)$? If not, find an inequality $\pi x \leq \pi_0$ satisfied by all points in X , but violated by the point x^* .

Now, in examining a problem to see if it has an efficient algorithm, we will see that the following four properties often go together:

(i) *Efficient Optimization Property*: For a given class of optimization problems $(P) \max\{cx : x \in X \subseteq R^n\}$, there exists an efficient (polynomial) algorithm.

(ii) *Strong Dual Property*: For the given problem class, there exists a strong dual problem $(D) \min\{\omega(z) : z \in U\}$ allowing us to obtain optimality conditions that can be quickly verified:

$x^* \in X$ is optimal in P if and only if there exists $u^* \in U$ with $cx^* = \omega(u^*)$.

(iii) *Efficient Separation Property*: There exists an efficient algorithm for the separation problem associated with the problem class.

(iv) *Explicit Convex Hull Property*: A compact description of the convex hull $\text{conv}(X)$ is known, which in principle allows us to replace every instance by the linear program: $\max\{cx : x \in \text{conv}(X)\}$.

Note that if a problem has the Explicit Convex Hull Property, then the dual of the linear program $\max\{cx : x \in \text{conv}(X)\}$ suggests that the Strong Dual Property should hold, and also using the description of $\text{conv}(X)$, there is some likelihood that the Efficient Separation Property holds. So some ties between the four properties are not surprising. The precise relationship will be discussed later. In the next sections we examine several classes of problems for which we will see that typically all four properties hold.

3.2 IPS WITH TOTALLY UNIMODULAR MATRICES

A natural starting point in solving integer programs :

$$(IP) \quad \max\{cx : Ax \leq b, x \in Z_+^n\}$$

with integral data (A, b) is to ask when one will be lucky, and the linear programming relaxation $(LP) \max\{cx : Ax \leq b, x \in R_+^n\}$ will have an optimal solution that is integral.

From linear programming theory, we know that basic feasible solutions take the form: $x = (x_B, x_N) = (B^{-1}b, 0)$ where B is an $m \times m$ nonsingular submatrix of (A, I) and I is an $m \times m$ identity matrix.

Observation 3.1 (Sufficient Condition) If the optimal basis B has $\det(B) = \pm 1$, then the linear programming relaxation solves IP .

Proof. From Cramer's rule, $B^{-1} = B^*/\det(B)$ where B^* is the adjoint matrix. The entries of B^* are all products of terms of B . Thus B^* is an integral matrix, and as $\det(B) = \pm 1$, B^{-1} is also integral. Thus $B^{-1}b$ is integral for all integral b . ■

The next step is to ask when one will always be lucky. When do all bases or all optimal bases satisfy $\det(B) = \pm 1$?

Definition 3.2 A matrix A is *totally unimodular (TU)* if every square submatrix of A has determinant $+1, -1$ or 0 .

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

Table 3.1 Matrices that are not TU

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

Table 3.2 Matrices that are TU

First we consider whether such matrices exist and how we can recognize them. Some simple observations follow directly from the definition.

Observation 3.2 If A is TU, $a_{ij} \in \{+1, -1, 0\}$ for all i, j .

Observation 3.3 The matrices in Table 3.1 are not TU. The matrices in Table 3.2 are TU.

Proposition 3.1 A matrix A is TU if and only if
(i) the transpose matrix A^T is TU if and only if
(ii) the matrix (A, I) is TU.

There is a simple and important sufficient condition for total unimodularity, that can be used to show that the first matrix in Table 3.2 is TU.

Proposition 3.2 (Sufficient Condition). A matrix A is TU if
(i) $a_{ij} \in \{+1, -1, 0\}$ for all i, j .
(ii) Each column contains at most two nonzero coefficients ($\sum_{i=1}^m |a_{ij}| \leq 2$).
(iii) There exists a partition (M_1, M_2) of the set M of rows such that each column j containing two nonzero coefficients satisfies $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$.

Proof. Assume that A is not TU, and let B be the smallest square submatrix of A for which $\det(B) \notin \{0, 1, -1\}$. B cannot contain a column with a single nonzero entry, as otherwise B would not be minimal. So B contains two nonzero entries in each column. Now by condition (iii), adding the rows in M_1 and subtracting the rows in M_2 gives the zero vector, and so $\det(B) = 0$, and we have a contradiction. ■

Note that condition (iii) means that if the nonzeros are in rows i and k , and if $a_{ij} = -a_{kj}$, then $\{i, k\} \in M_1$ or $\{i, k\} \in M_2$, whereas if $a_{ij} = a_{kj}$, $i \in M_1$

and $k \in M_2$, or vice versa. This leads to a simple algorithm to test whether the conditions of Proposition 3.2 hold. In the next section we will see an important class of matrices arising from network flow problems that satisfy this sufficient condition.

Now returning to IP, it is clear that when A is TU, the linear programming relaxation solves IP. In some sense the converse holds.

Proposition 3.3 *The linear program $\max\{cx : Ax \leq b, x \in R_+^n\}$ has an integral optimal solution for all integer vectors b for which it has a finite optimal value if and only if A is totally unimodular.*

On the question of efficient algorithms, we have essentially proved that for the IP: $\max\{cx : Ax \leq b, x \in Z_+^n\}$ with A totally unimodular:

- (a) The Strong Dual Property holds: the linear program $(D) : \min\{ub : uA \geq c, u \geq 0\}$ is a strong dual.
- (b) The Explicit Convex Hull Property holds: the convex hull of the set of feasible solutions $\text{conv}(X) = \{Ax \leq b, x \geq 0\}$ is known.
- (c) The Efficient Separation Property holds: the separation problem is easy as it suffices to check if $Ax^* \leq b$ and $x^* \geq 0$.

Given that these three properties hold, we have suggested that the Efficient Optimization Property should also hold, so there should be an efficient algorithm for IP. This turns out to be true, but it is a nontrivial result beyond the scope of this text. This is in turn related to the fact that efficient algorithms to recognize whether a matrix A is TU are also nontrivial.

3.3 MINIMUM COST NETWORK FLOWS

Here we consider an important class of problems with many applications lying at the frontier between linear and integer programming.

Given a digraph $D = (V, A)$ with arc capacities h_{ij} for all $(i, j) \in A$, demands b_i (positive inflows or negative outflows) at each node $i \in V$, and unit flow costs c_{ij} for all $(i, j) \in A$, the minimum cost network flow problem is to find a feasible flow that satisfies all the demands at minimum cost. This has the formulation:

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{3.1}$$

$$\sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = b_i \text{ for } i \in V \tag{3.2}$$

$$0 \leq x_{ij} \leq h_{ij} \text{ for } (i, j) \in A \tag{3.3}$$

where x_{ij} denotes the flow in arc (i, j) , $V^+(i) = \{k : (i, k) \in A\}$ and $V^-(i) = \{k : (k, i) \in A\}$.

It is evident that for the problem to be feasible the total sum of all the demands must be zero (i.e., $\sum_{i \in V} b_i = 0$).

Example 3.1 The digraph in Figure 3.1 leads to the following set of balance

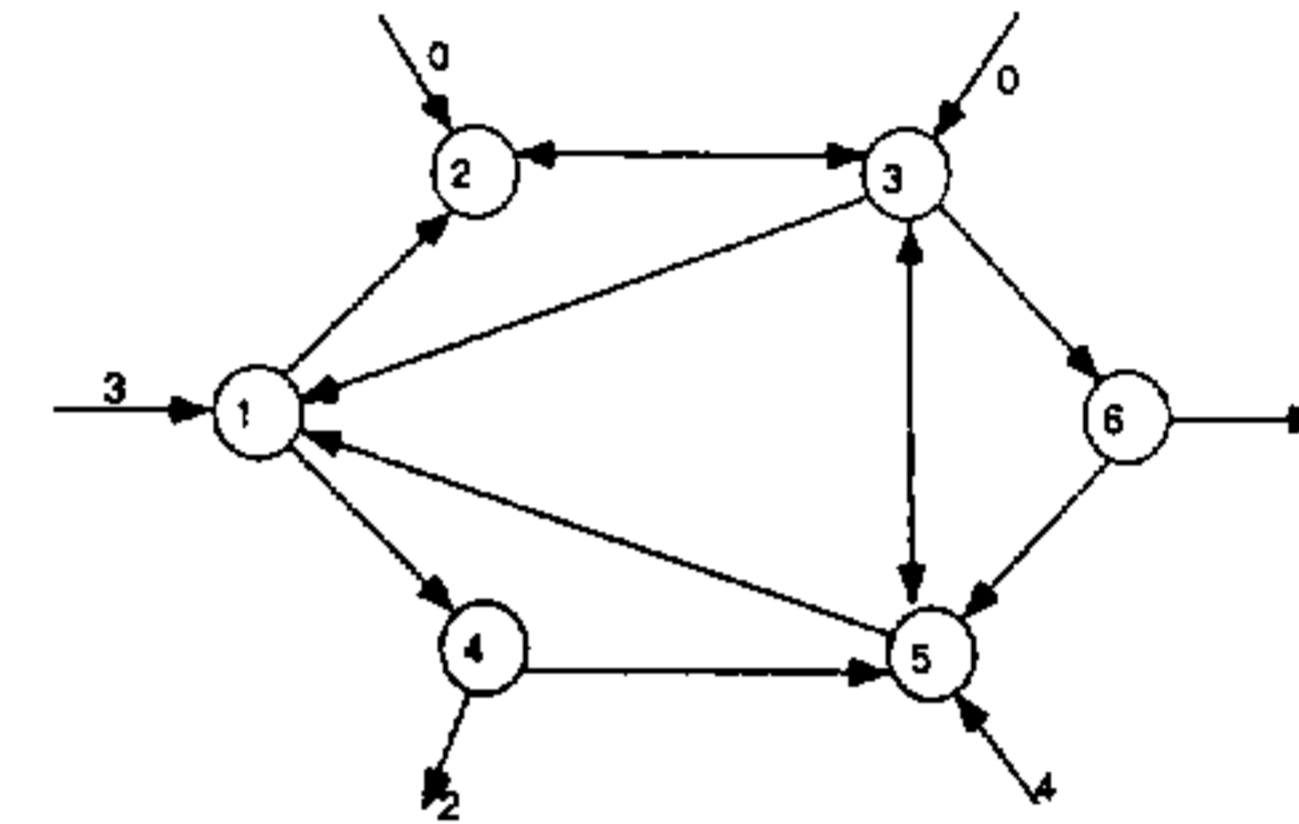


Fig. 3.1 Digraph for minimum cost network flow

equations:

x_{12}	x_{14}	x_{23}	x_{31}	x_{32}	x_{35}	x_{36}	x_{45}	x_{51}	x_{53}	x_{65}	
1	1	0	-1	0	0	0	0	-1	0	0	= 3
-1	0	1	0	-1	0	0	0	0	0	0	= 0
0	0	-1	1	1	1	1	0	0	-1	0	= 0
0	-1	0	0	0	0	0	1	0	0	0	= -2
0	0	0	0	0	-1	0	-1	1	1	-1	= 4
0	0	0	0	0	0	-1	0	0	0	1	= -5

The additional constraints are the bound constraints: $0 \leq x_{ij} \leq h_{ij}$. ■

Proposition 3.4 *The constraint matrix A arising in a minimum cost network flow problem is totally unimodular.*

Proof. The matrix A is of the form $\begin{pmatrix} C \\ I \end{pmatrix}$ where C comes from the flow conservation constraints, and I from the upper bound constraints. Therefore it suffices to show that C is TU. The sufficient conditions of Proposition 3.2 are satisfied with $M_1 = M$ and $M_2 = \phi$. ■

Corollary In a minimum cost network flow problem, if the demands $\{b_i\}$ and the capacities $\{h_{ij}\}$ are integral,

- (i) Each extreme point is integral.
- (ii) The constraints (3.2)-(3.3) describe the convex hull of the integral feasible flows.

3.4 SPECIAL MINIMUM COST FLOWS

The Shortest Path Problem. Given a digraph $D = (V, A)$, two distinguished nodes $s, t \in V$, and nonnegative arc costs c_{ij} for $(i, j) \in A$, find a minimum cost $s - t$ path.

The Max Flow Problem. Given a digraph $D = (V, A)$, two distinguished nodes $s, t \in V$, and nonnegative capacities h_{ij} for $(i, j) \in A$, find a maximum flow from s to t .

Both these problems are special cases of the minimum cost network flow problem so we can use total unimodularity to analyze them. However, the reader has probably already seen combinatorial polynomial algorithms for these two problems in a course on network flows.

What are the associated dual problems?

3.4.1 Shortest Path

Observe first that the shortest path problem can be formulated as:

$$z = \min \sum_{(i,j) \in A} c_{ij} x_{ij} \quad (3.4)$$

$$\sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = 1 \text{ for } i = s \quad (3.5)$$

$$\sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = 0 \text{ for } i \in V \setminus \{s, t\} \quad (3.6)$$

$$\sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} = -1 \text{ for } i = t \quad (3.7)$$

$$x_{ij} \geq 0 \text{ for } (i, j) \in A \quad (3.8)$$

$$x \in Z^{|A|} \quad (3.9)$$

where $x_{ij} = 1$ if arc (i, j) is in the minimum cost (shortest) $s - t$ path.

Theorem 3.5 z is the length of a shortest $s - t$ path if and only if there exist values π_i for $i \in V$ such that $\pi_s = 0$, $\pi_t = z$, and $\pi_j - \pi_i \leq c_{ij}$ for $(i, j) \in A$.

Proof. The linear programming dual of (3.4)–(3.8) is precisely

$$\begin{aligned} w^{LP} &= \max \pi_t - \pi_s \\ \pi_j - \pi_i &\leq c_{ij} \text{ for } (i, j) \in A. \end{aligned}$$

Replacing π_j by $\pi_j + \alpha$ for all $j \in V$ does not change the dual, so we can fix $\pi_s = 0$ without loss of generality. As the primal matrix is totally unimodular, strong duality holds and the claim follows. ■

We note that one particular dual solution is obtained by taking π_i to be the cost of a shortest path from s to i .

3.4.2 Maximum $s - t$ Flow

Adding a backward arc from t to s , the maximum $s - t$ flow problem can be formulated as:

$$\begin{aligned} \max x_{ts} \\ \sum_{k \in V^+(i)} x_{ik} - \sum_{k \in V^-(i)} x_{ki} &= 0 \text{ for } i \in V \\ 0 \leq x_{ij} &\leq h_{ij} \text{ for } (i, j) \in A. \end{aligned}$$

The dual is:

$$\begin{aligned} \min \sum_{(i,j) \in A} h_{ij} w_{ij} \\ u_i - u_j + w_{ij} &\geq 0 \text{ for } (i, j) \in A \\ u_t - u_s &\geq 1. \end{aligned}$$

From total unimodularity, an optimal solution is integer. Also as the dual is unchanged if we replace u_j by $u_j + \alpha$ for all $j \in V$, we can set $u_s = 0$. Given such a solution, let $X = \{j \in V : u_j \leq 0\}$ and $\bar{X} = V \setminus X = \{j \in V : u_j \geq 1\}$. Now

$$\sum_{(i,j) \in A} h_{ij} w_{ij} \geq \sum_{(i,j) \in A, i \in X, j \in \bar{X}} h_{ij} w_{ij} \geq \sum_{(i,j) \in A, i \in X, j \in \bar{X}} h_{ij}$$

as $w_{ij} \geq u_j - u_i \geq 1$ for $(i, j) \in A$ with $i \in X$ and $j \in \bar{X}$.

However, this lower bound $\sum_{(i,j) \in A, i \in X, j \in \bar{X}} h_{ij}$ is attained by the solution $u_j = 0$ for $j \in X$, $u_j = 1$ for $j \in \bar{X}$, $w_{ij} = 1$ for $(i, j) \in A$ with $i \in X$ and $j \in \bar{X}$, and $w_{ij} = 0$ otherwise. So there is an optimal 0-1 solution.

We see that $s \in X, t \in \bar{X}$, $\{(i, j) : w_{ij} = 1\}$ is the set of arcs of the $s - t$ cut $(X, V \setminus X)$, and we obtain the standard result that the maximum value of an $s - t$ flow equals the minimum capacity of an $s - t$ cut.

Theorem 3.6 A strong dual to the max $s - t$ flow problem is the minimum $s - t$ cut problem:

$$\min_X \left\{ \sum_{(i,j) \in A, i \in X, j \notin X} h_{ij} : s \in X \subset V \setminus \{t\} \right\}.$$

3.5 OPTIMAL TREES

Definition 3.3 Given a graph $G = (V, E)$, a *forest* is a subgraph $G' = (V, E')$ containing no cycles.

Definition 3.4 Given a graph $G = (V, E)$, a *tree* is a subgraph $G' = (V, E')$ that is a forest and is *connected* (contains a path between every pair of nodes of V).

Some well-known consequences of these definitions are listed below:

- Proposition 3.7** A graph $G = (V, E)$ is a tree if and only if
- (i) it is a forest containing exactly $n - 1$ edges, if and only if
 - (ii) it is an edge-minimal connected graph spanning V , if and only if
 - (iii) it contains a unique path between every pair of nodes of V , if and only if
 - (iv) the addition of an edge not in E creates a unique cycle.

The Maximum Weight Forest (Tree) Problem. Given a graph $G = (V, E)$ and edge weights c_e for $e \in E$, find a maximum weight subgraph that is a forest (tree).

This problem arises naturally in many telecommunications and computer network applications where it is necessary that there is at least one path between each pair of nodes. When one wishes to minimize installation costs, the optimal solution is clearly a minimum weight tree.

Remember from the previous chapter that the idea of a *greedy algorithm* is to take the best element and run. It is very shortsighted. It just chooses one after the other whichever element gives the maximum profit and still gives a feasible solution. For a graph $G = (V, E)$, we let $n = |V|$ and $m = |E|$.

Greedy Algorithm for a Maximum Weight Tree

Initialization. Set $E^0 = E, T^0 = \phi$. Order the edges by nonincreasing weight $c_1 \geq c_2 \geq \dots \geq c_m$, where c_t is the cost of edge e_t .

Iteration t . If $T^{t-1} \cup \{e_t\}$ contains no cycle, set $T^t = T^{t-1} \cup \{e_t\}$. Otherwise $T^t = T^{t-1}$.

Set $E^t = E^{t-1} \setminus \{e_t\}$. If $|T^t| = n - 1$, stop with T^t is optimal. If $t = m$, stop with no feasible solution.

To obtain a maximum weight forest, it suffices to modify the greedy algorithm to stop as soon as $c_{t+1} \leq 0$.

Example 3.2 Consider the graph shown in Figure 3.2 with the weights on the edges as shown.

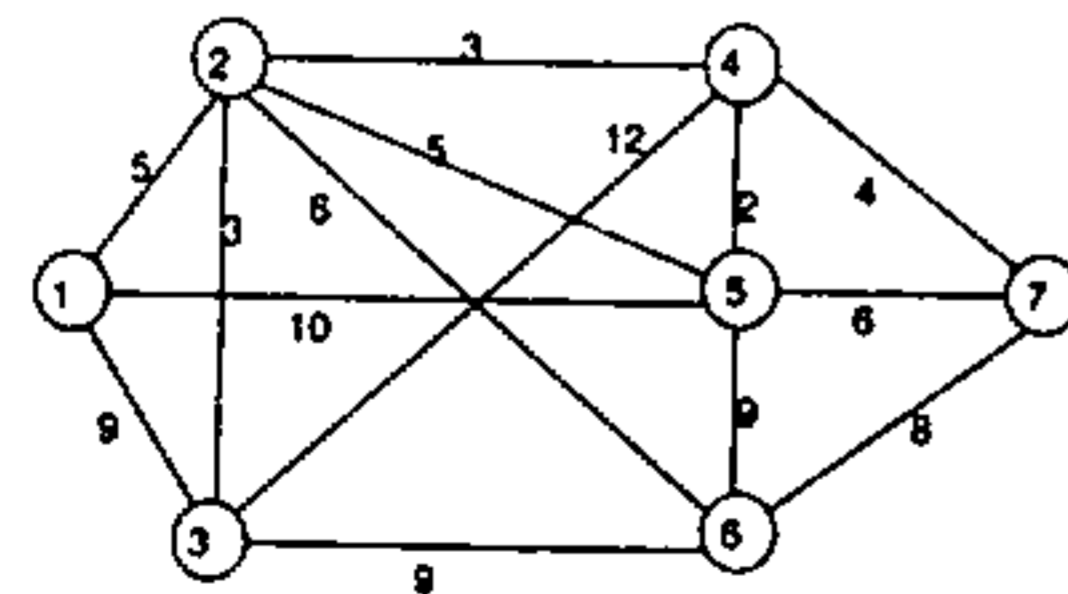


Fig. 3.2 Graph for optimal weight tree

We consider the edges in the order:

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
(i, j)	(3, 4)	(1, 5)	(1, 3)	(3, 6)	(5, 6)	(6, 7)	(5, 7)
c_e	12	10	9	9	9	8	6
	+	+	+	+		+	
	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}
	(2, 6)	(1, 2)	(2, 5)	(4, 7)	(2, 3)	(2, 4)	(4, 5)
	6	5	5	4	3	3	2
	+						

The algorithm chooses the edges marked with a + and rejects the others. For example, the first edge rejected is $e_5 = (5, 6)$ because it forms a cycle with e_2, e_3 , and e_4 , which have already been selected. ■

Theorem 3.8 The greedy algorithm terminates with an optimal weight tree.

Proof. Suppose for simplicity that all the edge weights are different. Let $T = \{g_1, \dots, g_{n-1}\}$ be the edges chosen by the greedy algorithm with $c_{g_1} > \dots > c_{g_{n-1}}$. Let $F = \{f_1, \dots, f_{n-1}\}$ be the edges of an optimal solution with $c_{f_1} > \dots > c_{f_{n-1}}$.

If the solutions are the same, the result is proved. So suppose the two solutions differ with $g_1 = f_1, \dots, g_{k-1} = f_{k-1}$ but $g_k \neq f_k$.

- (i) We observe that $c_{g_k} > c_{f_k}$ because the greedy algorithm chooses g_k and not f_k and neither edge creates a cycle with $\{g_1, \dots, g_{k-1}\}$. Also by construction $c_{f_1} > \dots > c_{f_{n-1}}$, and so $g_k \notin F$.
- (ii) Now consider the edge set $F \cup \{g_k\}$. As F is a tree, it follows from Proposition 3.7 that $F \cup \{g_k\}$ contains exactly one cycle C . Note however that the set of edges $\{f_1, \dots, f_{k-1}, g_k\} = \{g_1, \dots, g_{k-1}, g_k\}$ forms part of the tree T and thus does not contain a cycle. Therefore one of the other edges f_k, \dots, f_{n-1} must be in the cycle C . Suppose f^* is one such edge.
- (iii) As C contains a unique cycle and f^* is in this cycle, $T' = F \cup \{g_k\} \setminus \{f^*\}$ is cycle free. As it has $n - 1$ edges, it is a tree.
- (iv) Finally, as $c_{g_k} > c_{f_k}$ and $c_{f_k} \geq c_{f^*}$, the weight of T' exceeds that of F .
- (v) As F is an optimal tree, we have arrived at a contradiction, and so T and F cannot differ. ■

As the greedy algorithm is easily seen to be polynomial, the Efficient Optimization Property holds for the maximum weight tree problem. So we can again consider the other three properties. To do this, we need a formulation of the problem as an integer program. In modeling the traveling salesman problem in Section 1.2 we saw how to avoid cycles. Thus the maximum weight forest problem can be formulated as:

$$\max \sum_{e \in E} c_e x_e \tag{3.10}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \text{ for } 2 \leq |S| \leq n, S \subseteq V \tag{3.11}$$

$$x_e \geq 0 \text{ for } e \in E \quad (3.12)$$

$$x \in Z^{|E|}. \quad (3.13)$$

Theorem 3.9 *The convex hull of the incidence vectors of the forests in a graph is given by the constraints (3.11)–(3.12).*

This result says that the Explicit Convex Hull Property holds for the Maximum Weight Forest Problem. We generalize and prove this result in the next section, and in Chapter 9 we show that the Efficient Separation Property holds for this problem.

To terminate this section we introduce an important and more difficult generalization of the optimal tree problem. Given a graph $G = (V, E)$ and a set of terminals $T \subseteq V$, a *Steiner tree on T* is an edge-minimal acyclic subgraph of G containing a path joining every pair of nodes in T . Such a subgraph may or may not have edges incident to the nodes in $V \setminus T$. Given weights c_e for $e \in E$, the *Optimal Steiner Tree Problem* is to find a minimum weight Steiner tree. Observe that when $T = V$, this is the optimal tree problem, and when $|T| = 2$, it is the shortest path problem.

3.6 SUBMODULARITY AND MATROIDS*

Here we examine a larger class of problems for which a greedy algorithm provides an optimal solution. This generalizes the maximum weight forest problem examined in the last section. $\mathcal{P}(N)$ denotes the set of subsets of N .

Definition 3.5 (i) A set function $f: \mathcal{P}(N) \rightarrow R^1$ is *submodular* if

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B) \text{ for all } A, B \subseteq N.$$

(ii) A set function f is *nondecreasing* if

$$f(A) \leq f(B) \text{ for all } A, B \text{ with } A \subset B \subseteq N.$$

An alternative representation of such functions is useful.

Proposition 3.10 *A set function f is non-decreasing and submodular if and only if*

$$f(A) \leq f(B) + \sum_{j \in A \setminus B} [f(B \cup \{j\}) - f(B)] \text{ for all } A, B \subseteq N.$$

Proof. Suppose f is non-decreasing and submodular. Let $A \setminus B = \{j_1, \dots, j_r\}$. Then $f(A) - f(B) \leq f(A \cup B) - f(B) = \sum_{i=1}^r [f(B \cup \{j_1, \dots, j_i\}) -$

$f(B \cup \{j_1, \dots, j_{i-1}\})] \leq \sum_{i=1}^r [f(B \cup \{j_i\}) - f(B)] = \sum_{j \in A \setminus B} [f(B \cup \{j\}) - f(B)]$, where the first inequality follows from f non-decreasing and the second from submodularity. The other direction is immediate. ■

Now given a nondecreasing submodular function f on N with $f(\emptyset) = 0$, we consider the *submodular polyhedron*:

$$P(f) = \{x \in R_+^n : \sum_{j \in S} x_j \leq f(S) \text{ for } S \subseteq N\},$$

and the associated *submodular optimization problem*:

$$\max\{cx : x \in P(f)\}.$$

The Greedy Algorithm for the Submodular Optimization Problem.

- (i) Order the variables so that $c_1 \geq c_2 \geq \dots \geq c_r > 0 \geq c_{r+1} \geq \dots \geq c_n$.
- (ii) Set $x_i = f(S^i) - f(S^{i-1})$ for $i = 1, \dots, r$ and $x_j = 0$ for $j > r$, where $S^i = \{1, \dots, i\}$ for $i = 1, \dots, r$ and $S^0 = \emptyset$.

Theorem 3.11 *The greedy algorithm solves the submodular optimization problem.*

Proof. As f is nondecreasing, $x_i = f(S^i) - f(S^{i-1}) \geq 0$ for $i = 1, \dots, r$. Also for each $T \subseteq N$,

$$\begin{aligned} \sum_{j \in T} x_j &= \sum_{j \in T \cap S^r} [f(S^j) - f(S^{j-1})] \\ &\leq \sum_{j \in T \cap S^r} [f(S^j \cap T) - f(S^{j-1} \cap T)] \\ &\leq \sum_{j \in S^r} [f(S^j \cap T) - f(S^{j-1} \cap T)] \\ &= f(S^r \cap T) - f(\emptyset) \leq f(T), \end{aligned}$$

where the first inequality follows from the submodularity of f , and the others as f is nondecreasing. So the greedy solution is feasible with value $\sum_{i=1}^r c_i [f(S^i) - f(S^{i-1})]$.

Now consider the linear programming dual:

$$\begin{aligned} \min \sum_{S \subseteq N} f(S) y_S \\ \sum_{S: j \in S} y_S \geq c_j \text{ for } j \in N \\ y_S \geq 0 \text{ for } S \subseteq N. \end{aligned}$$

Let $y_{S^i} = c_i - c_{i+1}$ for $i = 1, \dots, r-1$, $y_{S^r} = c_r$, and $y_S = 0$ otherwise. Clearly $y_S \geq 0$ for all $S \subseteq N$. Also for $j \leq r$, $\sum_{S: j \in S} y_S \geq \sum_{i=j}^r y_{S^i} =$

$\sum_{i=1}^{r-1} (c_i - c_{i+1}) + c_r = c_j$, and for $j > r$, $\sum_{S: j \in S} y_S \geq 0 \geq c_j$. Thus the solution y is dual feasible. Finally the dual objective value is

$$\sum_{i=1}^r f(S^i) y_{S^i} = \sum_{i=1}^{r-1} f(S^i) (c_i - c_{i+1}) + f(S^r) c_r = \sum_{i=1}^r c_i [f(S^i) - f(S^{i-1})].$$

So the value of the dual feasible solution has the same value as the greedy solution, and so from linear programming duality, the greedy solution is optimal. ■

Note that when f is integer-valued, the greedy algorithm provides an integral solution. In the special case when $f(S \cup \{j\}) - f(S) \in \{0, 1\}$ for all $S \subset N$ and $j \in N \setminus S$, we call f a *submodular rank function*, and the greedy solution is a 0-1 vector. What is more, we now show that the feasible 0-1 points in the submodular rank polyhedron generate an interesting combinatorial structure, called a *matroid*.

Proposition 3.12 Suppose that r is a submodular rank function on a set N with $r(\emptyset) = 0$.

- (i) $r(A) \leq |A|$ for all $A \subseteq N$.
- (ii) If $r(A) = |A|$, then $r(B) = |B|$ for all $B \subset A \subseteq N$.
- (iii) If x^A is the incidence vector of $A \subseteq N$, $x^A \in P(r)$ if and only if $r(A) = |A|$.

Proof. (i) Using Proposition 3.10 and the property of a submodular rank function, $r(A) \leq r(\emptyset) + \sum_{j \in A} [r(\{j\}) - r(\emptyset)] \leq |A|$.

(ii) Again using the same properties, $|A| = r(A) \leq r(B) + \sum_{j \in A \setminus B} [r(B \cup \{j\}) - r(B)] \leq |B| + |A \setminus B| = |A|$. Equality must hold throughout, and thus $r(B) = |B|$.

(iii) If $r(A) < |A|$, $\sum_{j \in A} x_j^A = |A| > r(A)$ and $x^A \notin P(r)$. If $r(A) = |A|$, $\sum_{j \in S} x_j^A = |A \cap S| = r(A \cap S) \leq r(S)$ where the second equality uses (ii). This inequality holds for all $S \subseteq N$, and thus $x^A \in P(r)$. ■

Definition 3.6 Given a submodular rank function r , a set $A \subseteq N$ is *independent* if $r(A) = |A|$. The pair (N, \mathcal{F}) , where \mathcal{F} is the set of independent sets, is called a *matroid*.

Based on Theorem 3.11, we know how to optimize on matroids.

Theorem 3.13 The greedy algorithm solves the maximum weight independent set problem in a matroid.

Given a connected graph $G = (V, E)$, it is not difficult to verify that the edge sets of forests form a matroid and that the function $r : \mathcal{P}(E) \rightarrow R^1$, where $r(E')$ is the size of the largest forest in (V, E') , is a submodular rank function. Specifically when $S \subseteq V$, $E' = E(S)$ and the subgraph $(S, E(S))$

is connected, we clearly have $r(E(S)) = |S| - 1$, so the forest polyhedron (3.11)-(3.12) is a special case of a submodular polyhedron.

The constraint set associated to a submodular polyhedron has another interesting property.

Definition 3.7 A set of linear inequalities $Ax \leq b$ is called *Totally Dual Integral (TDI)* if, for all $c \in Z^n$ for which the linear program $\max\{cx : Ax \leq b\}$ has a finite optimal value, the dual linear program

$$\min\{yb : yA = c, y \geq 0\}$$

has an optimal solution with y integral.

We have seen in the proof of Theorem 3.11 that the linear system $\{\sum_{j \in S} x_j \leq f(S) \text{ for } S \subseteq N, x_j \geq 0 \text{ for } j \in N\}$ is TDI. Based on the following result, the TDI property provides another useful way of showing that certain linear programs always have integer solutions.

Theorem 3.14 If $Ax \leq b$ is TDI, b is an integer vector, and $P = \{x \in R^n : Ax \leq b\}$ has vertices, then all vertices of P are integral.

Note also that if A is a TU matrix, then $Ax \leq b$ is TDI.

3.7 NOTES

3.1 The theoretical importance of the separation problem is discussed at the end of Chapter 6. Its practical importance was brought out in the first computational studies using strong cutting planes; see Chapter 9.

3.2 Totally unimodular matrices have been studied since the fifties. The characterization of Proposition 3.1 is due to [HofKru56]. The interval or consecutive 1's property of Exercise 3.3 is due to [FulGro65], and the stronger necessary condition is from [Gho62]. A complete characterization of TU matrices is much more difficult; see [Sey80] or the presentation in [Sch86].

3.3 We again refer to [AhuMagOrl93] for network flows, as well as shortest path and max flow problems. This book also contains a large number of applications and a wealth of exercises. Note also the chapter of Ahuja on flows and paths in [DelAMafMar97], which also indicates where some of the latest software for network flow problems can be obtained.

3.4 The max flow min cut theorem was already part of the max flow algorithm of [ForFul56]. The min cut problem arises as a separation problem in solving TSP and other network design problems. The problem of finding all minimum cuts was answered in [GomHu61]. Recently new algorithms have appeared that find minimum cuts directly without using flows; see Ch. 3 in

[CooCunetal97].

3.5 The greedy algorithm for finding minimum weight trees is from [Kru56]. A faster classical algorithm is that of [Prim57]. Special algorithms based on Delaunay triangulations can be used for two-dimensional Euclidean problems, [PreSha85]. [Goe94] and [MagWol95] contain a discussion of many alternative formulations for tree and Steiner tree problems.

3.6 Submodular polyhedra and the greedy algorithm for matroids are studied in [Edm70] and [Edm71], see also [Law76]. [Wel76] is a book devoted to matroids. For total dual integrality, see [EdmGil77].

3.8 EXERCISES

1. Are the following matrices totally unimodular or not?

$$A_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}, A_2 = \begin{pmatrix} -1 & 1 & -1 & & \\ & 1 & 1 & 1 & 1 \\ -1 & 1 & & & \\ & & 1 & & 1 & 1 \\ & & & 1 & -1 & \end{pmatrix}.$$

2. Prove that the polyhedron $P = \{(x_1, \dots, x_m, y) \in R_+^{m+1} : y \leq 1, x_i \leq y \text{ for } i = 1, \dots, m\}$ has integer vertices.

3. A 0-1 matrix B has the *consecutive 1's property* if for any column j , $b_{ij} = b_{i'j} = 1$ with $i < i'$ implies $b_{lj} = 1$ for $i < l < i'$.

A more general sufficient condition for total unimodularity is: Matrix A is TU if

- (i) $a_{ij} \in \{+1, -1, 0\}$ for all i, j .
- (ii) For any subset M of the rows, there exists a partition (M_1, M_2) of M such that each column j satisfies

$$\left| \sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} \right| \leq 1.$$

Use this to show that a matrix with the consecutive 1's property is TU.

4. Consider a scheduling model in which a machine can be switched on at most k times: $\sum_t z_t \leq k, z_t - y_t + y_{t-1} \geq 0, z_t \leq y_t, 0 \leq y_t, z_t \leq 1$ for all t , where $y_t = 1$ if the machine is on in period t , and $z_t = 1$ if it is switched on in period t . Show that the resulting matrix is TU.

5. Prove Proposition 3.3.

6. Use linear programming to find the length of a shortest path from node s to node t in the directed graph of Figure 3.3. Use an optimal dual solution to prove that your solution is optimal.

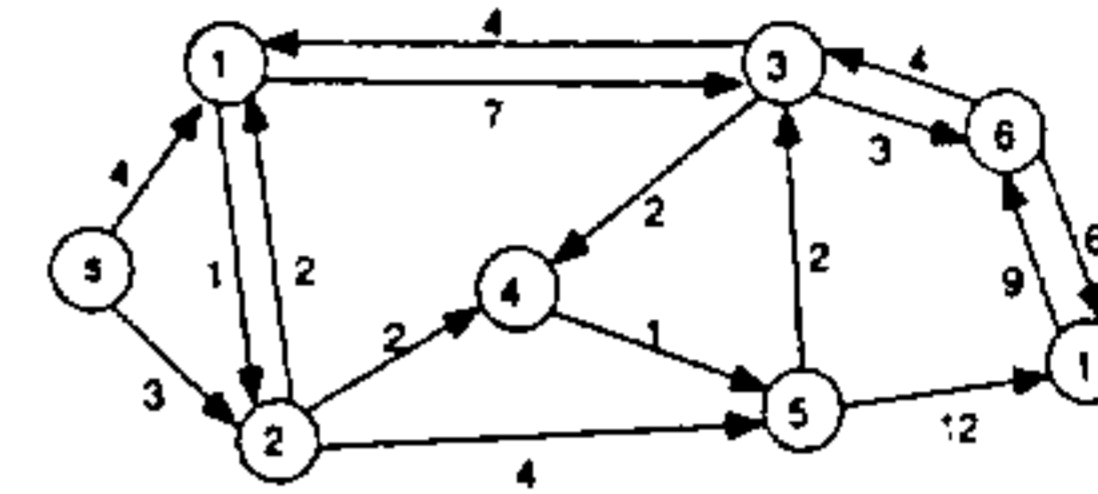


Fig. 3.3 Shortest path instance

7. Use linear programming to find a minimum $s - t$ cut in the capacitated network of Figure 3.4.

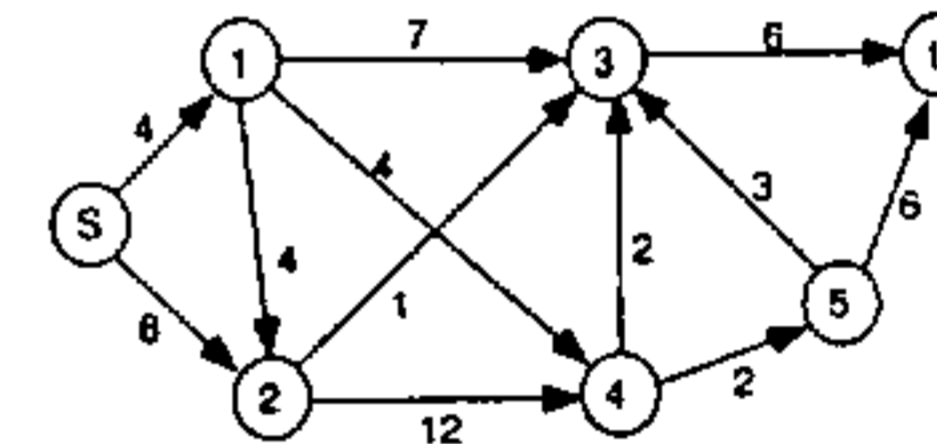


Fig. 3.4 Network instance

8. Find a minimum weight spanning tree in the graph shown in Figure 3.5.

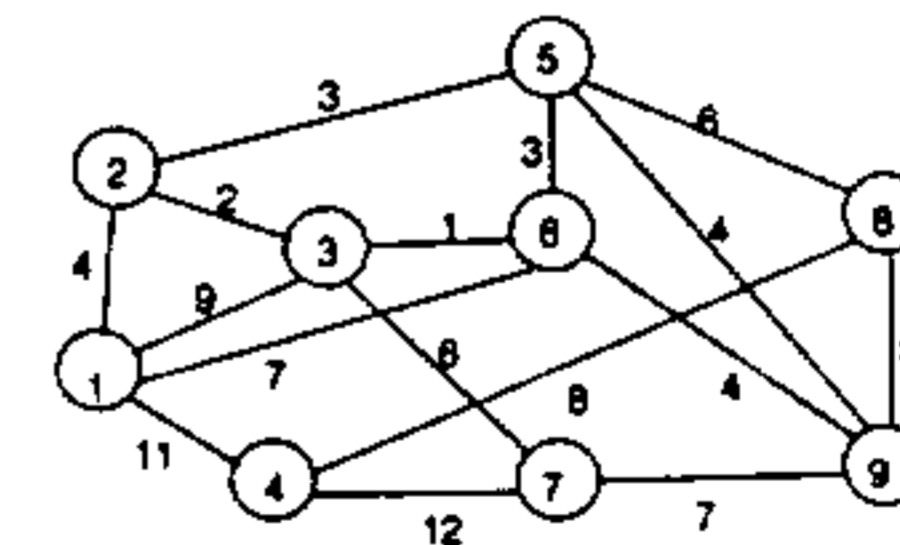


Fig. 3.5 Tree instance

9. Prove that the greedy algorithm produces an optimal weight tree when edge weights can be equal.

10. Formulate the optimal Steiner tree problem as an integer program.

11. (i) For their annual Christmas party the thirty members of staff of the thriving company Ipopt were invited/obliged to dine together and then spend the night in a fancy hotel. The boss's secretary had the unenviable task of allocating the staff two to a room. Knowing the likes and dislikes of everyone, she drew up a list of all the compatible pairs. How could you help her to fill all fifteen rooms?

(ii) Recently a summer camp was organized for an equal number of English and French children. After a few days, the children had to participate in an orienteering competition in pairs, each pair made up of one French and one English child. To allocate the pairs, each potential pair was asked to give a weight from 1 to 10 representing their willingness to form a pair. Formulate the problem of choosing the pairs so as to maximize the sum of the weights.

(iii) If you have a linear programming code available, can you help either the boss's secretary or the camp organizer or both?

12. Consider a real matrix C with n columns. Let $N = \{1, \dots, n\}$ and $\mathcal{F} = \{S \subseteq N : \text{the columns } \{c_j\}_{j \in S} \text{ are linearly independent}\}$. Show that (N, \mathcal{F}) is a matroid. What is the associated rank function r ?

13. Given a matroid, show that

(i) if A and B are independent sets with $|A| > |B|$, then there exists $j \in A \setminus B$ such that $A \cup \{j\}$ is independent, and

(ii) for an arbitrary set $A \subseteq N$, every maximal independent set in A has the same cardinality.

4

Matchings and Assignments

4.1 AUGMENTING PATHS AND OPTIMALITY

Here we demonstrate two other important ideas used in certain combinatorial algorithms. One idea is that of a primal algorithm systematically moving from one feasible solution to a better one. The second is that of iterating between primal and dual problems using the LP complementarity conditions.

First a few reminders. We suppose that a graph $G = (V, E)$ is given.

Definition 4.1 A *matching* $M \subseteq E$ is a set of disjoint edges, that is, at most one edge of a matching is incident to any node $v \in V$.

Definition 4.2 A *covering by nodes* is a set of nodes $R \subseteq V$ such that every edge $e \in E$ is incident to at least one of the nodes of R .

We have shown in Section 2.5 that there is a weak duality between matchings and coverings by nodes, namely for every matching M and covering by nodes R , $|M| \leq |R|$. Here we consider the *Maximum Cardinality Matching Problem* $\max\{|M| : M \text{ is a matching}\}$, and to solve it we examine first how to construct matchings of larger and larger cardinality.

Definition 4.3 An *alternating path* with respect to a matching M is a path $P = v_0, e_1, v_1, e_2, \dots, e_p, v_p$ such that

(i) $e_1, e_3, \dots, e_{\text{odd}} \in E \setminus M$.

(ii) $e_2, e_4, \dots, e_{\text{even}} \in M$.

(iii) v_0 is not incident to the matching M (v_0 is an *exposed* node).

An *augmenting path* is an alternating path that in addition satisfies the condition:

(iv) The number of edges p is odd, and v_p is not incident to the matching M .

Augmenting paths are what we need (see Figure 4.1).

Proposition 4.1 *Given a matching M and an augmenting path P relative to M , the symmetric difference $M' = (M \cup P) \setminus (M \cap P)$ is a matching with $|M'| > |M|$.*

Proof. As v_1 and v_p do not touch M , M' is a matching. As p is odd, $|P \cap (E \setminus M)| = |P \cap M| + 1$. Thus $|M'| = |M| + 1$. ■

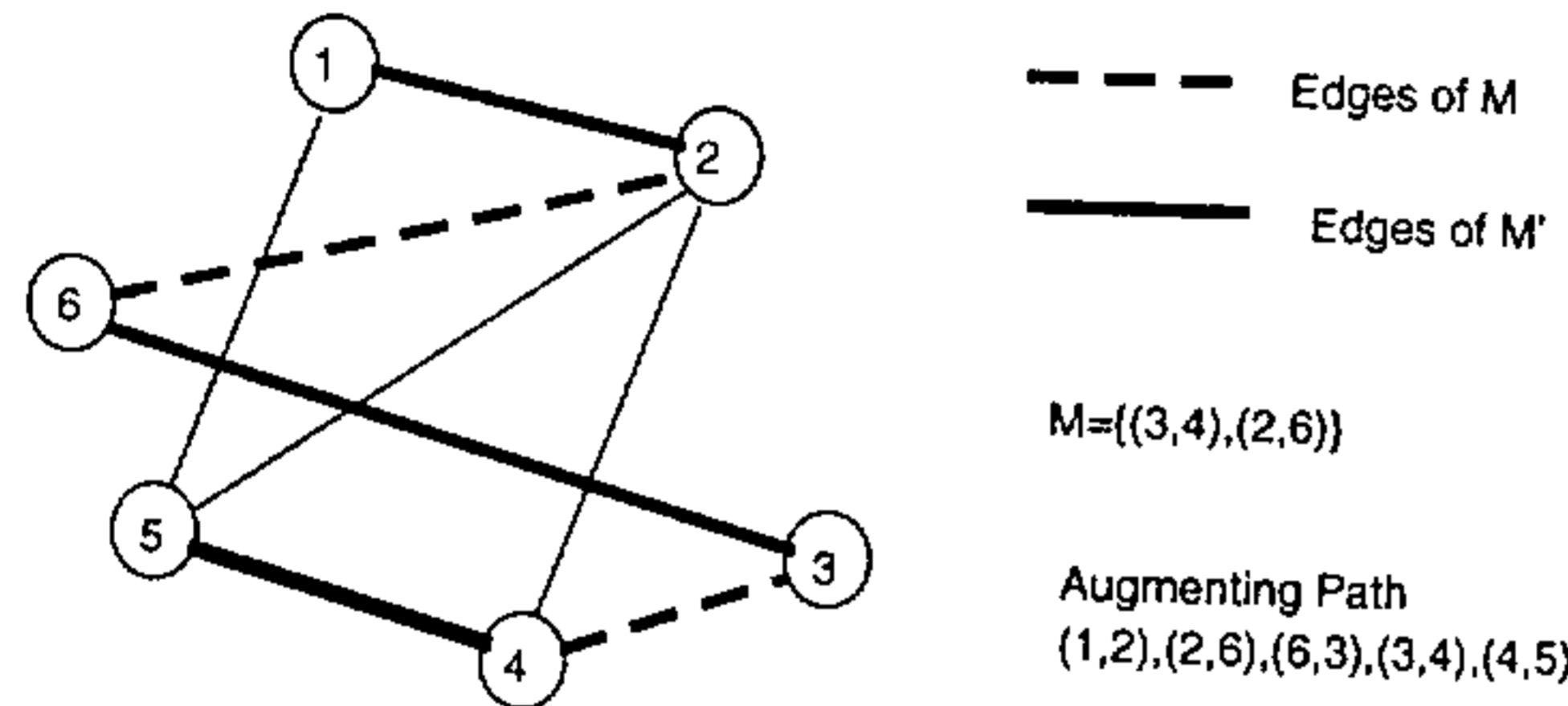


Fig. 4.1 An augmenting path

So the existence of an augmenting path implies that M is not optimal. Is the converse also true? If there is no augmenting path, can we conclude that M is optimal?

Proposition 4.2 *If there is no augmenting path relative to a matching M , then M is of maximum cardinality.*

Proof. We show the contrapositive. We suppose that M is not optimal. Thus there exists a matching M' with $|M'| > |M|$. Consider the graph whose edges are given by $(M \cup M') \setminus (M \cap M')$. The degree of each node in this graph is 0, 1, or 2. Thus the connected components of the graph are paths and cycles. For a cycle C , the edges alternate between M and M' , and so the cycles are of even length and contain the same number of edges from M and M' . The paths can contain either an even or an odd number of edges. As $|M'| > |M|$, one of the paths must contain more edges from M' than from M . This path is an augmenting path. ■

Can we find whether there is an augmenting path or not in polynomial time? If there is no augmenting path, can we give a simple way to verify that

the matching we have is maximum? In the next section we give a positive answer to both these questions when the graph is bipartite.

4.2 BIPARTITE MAXIMUM CARDINALITY MATCHING

Given a bipartite graph $G = (V_1, V_2, E)$, where $V = V_1 \cup V_2$ and every edge has one endpoint in V_1 and the other in V_2 , we wish to find a maximum size matching in G . We suppose that a matching M (possibly empty) has been found, and we wish either to find an augmenting path, or to demonstrate that there is no such path and that the matching M is optimal.

We try to systematically examine all augmenting paths.

Observation 4.1 As augmenting paths P are of odd length and the graph is bipartite, one of the exposed nodes of P is in V_1 and the other in V_2 . Thus it suffices to start enumerating from V_1 .

Outline of the Algorithm. We start by labeling all the nodes of V_1 disjoint from M . These are candidates to be the first node of an alternating path.

The first (and subsequent odd) edges of an alternating path are in $E \setminus M$, and thus all such edges from the labeled nodes in V_1 are candidates. The endpoints of these edges in V_2 are then labeled.

The second (and subsequent even) edges of an alternating path are in M , and thus any edge in M touching a labeled node in V_2 is a candidate. The endpoints of these edges in V_1 are then labeled, and so on.

The labeling stops: either when a node of V_2 is labeled that is not incident to M , so an augmenting path has been found, or when no more edges can be labeled, and so none of the alternating paths can be extended further.

Algorithm for Bipartite Maximum Cardinality Matching

Step 0. $G = (V_1, V_2, E)$ is given. M is a matching. No nodes are labeled or scanned.

Step 1. (Labeling)

1.0 Give the label $*$ to each exposed node in V_1 .

1.1 If there are no unscanned labels, go to Step 3. Choose a labeled unscanned node i . If $i \in V_1$, go to 1.2. If $i \in V_2$, go to 1.3.

1.2 Scan the labeled node $i \in V_1$. For all $(i, j) \in E \setminus M$, give j the label i if j is unlabeled. Return to 1.1.

1.3 Scan the labeled node $i \in V_2$. If i is exposed, go to Step 2. Otherwise, find the edge $(j, i) \in M$ and give node $j \in V_1$ the label i . Return to 1.1.

Step 2 (Augmentation). An augmenting path P has been found. Use the labels to backtrack from $j \in V_2$ to find the path.

Augment M . $M \leftarrow (M \cup P) \setminus (M \cap P)$. Remove all labels. Return to Step 1.

Step 3 (No Augmenting Path): Let V_1^+, V_2^+ be the nodes of V_1 and V_2 that are labeled and V_1^-, V_2^- the unlabeled nodes.

Theorem 4.3 On termination of the algorithm,

- (i) $R = V_1^- \cup V_2^+$ is a node covering of the edges E of G .
- (ii) $|M| = |R|$, and M is optimal.

Proof. (a) As no more nodes can be labeled, from Step 1.2 it follows that there is no edge from V_1^+ to V_2^- . This means that $V_1^- \cup V_2^+$ covers E .

(b) As no augmenting path is found, every node of V_2^+ is incident to an edge e of M , and from Step 1.3 the other endpoint is in V_1^+ .

(c) Every node of V_1^- is incident to an edge e of M , as otherwise it would have received the label $*$ in Step 1.0. The other endpoint is necessarily in V_2^- , as otherwise the node would have been labeled in Step 1.2.

(d) Thus $|V_1^- \cup V_2^+| \leq |M|$. But $|R| \geq |M|$ and thus $|R| = |M|$. ■

Example 4.1 Consider the bipartite graph shown in Figure 4.2 and the initial matching $M = \{(3, 8), (5, 10)\}$. The algorithm leads to the labeling shown, and the construction of the set of alternating paths shown. Two alternating paths are found: $(1, 8), (3, 8), (3, 7)$ and $(4, 10), (5, 10), (5, 9)$.

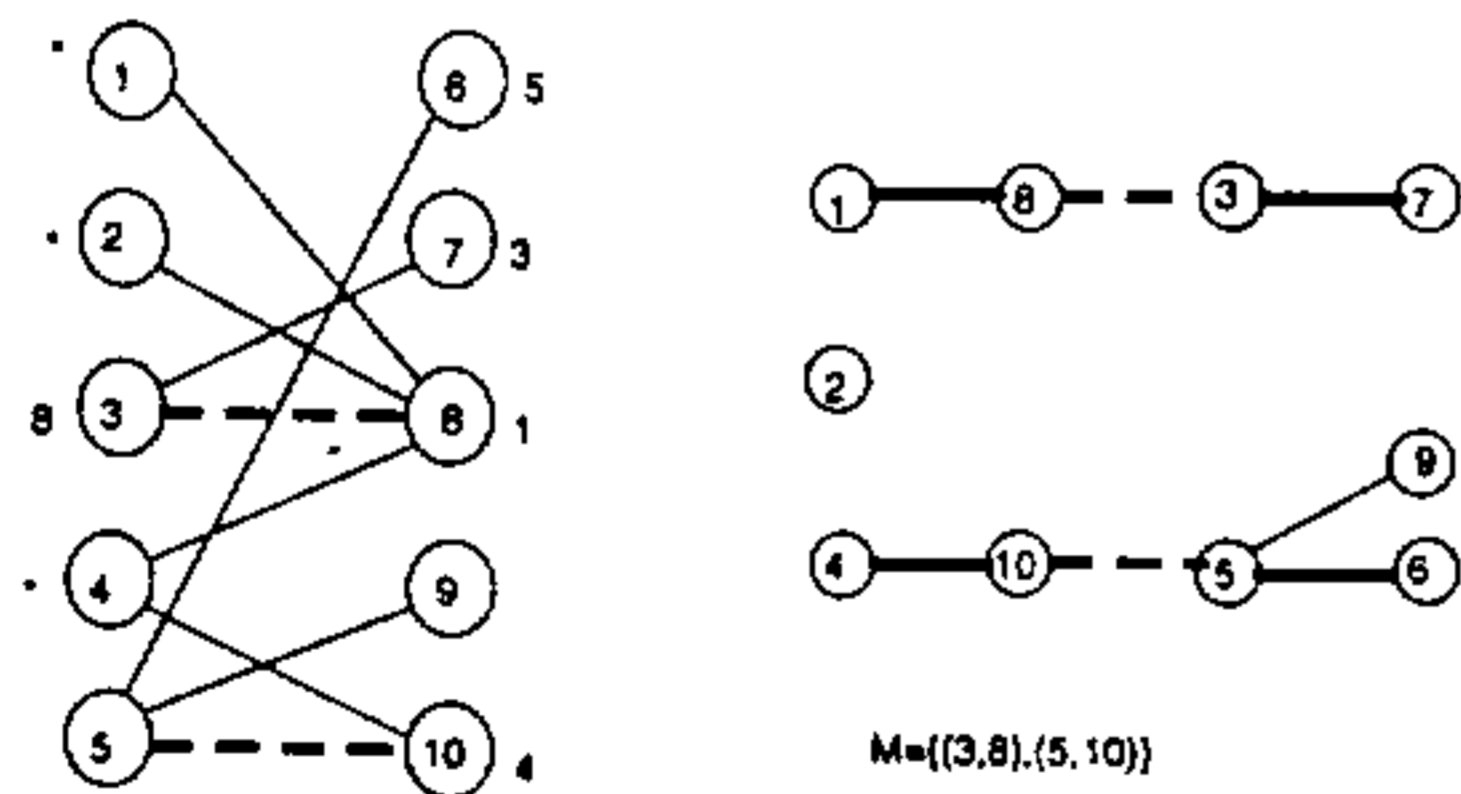


Fig. 4.2 Bipartite matching

In Figure 4.3 we show the new matching $M = \{(1, 8), (3, 7), (4, 10), (5, 9)\}$ and the labeling obtained from the algorithm. Now we see that we cannot add any more labels and no augmenting path has been found. It is easily checked that the node set $R = \{3, 4, 5, 8\}$ is an edge cover. As $|M| = |R| = 4$, M is optimal. ■

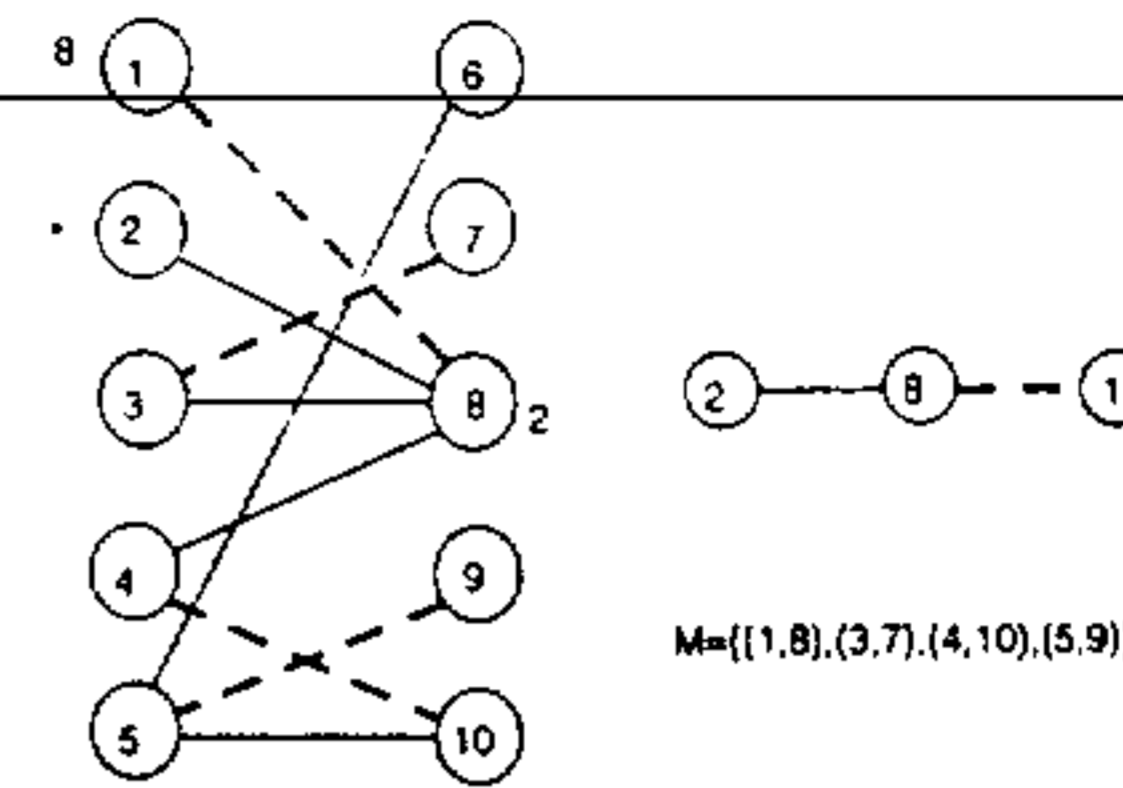


Fig. 4.3 Bipartite matching 2

4.3 THE ASSIGNMENT PROBLEM

Given a bipartite graph $G = (V_1, V_2, E)$ and weights c_e for $e \in E$, one problem is to find a matching of maximum weight. From the results on total unimodularity, we know that it suffices to solve the linear program:

$$z = \max \sum_{e \in E} c_e x_e$$

$$\sum_{e \in \delta(i)} x_e \leq 1 \text{ for } i \in V_1 \cup V_2$$

$$x_e \geq 0 \text{ for } e \in E.$$

When $|V_1| = |V_2| = n$, and the problem is to find a matching of size n of maximum weight, we obtain the assignment problem:

$$z = \max \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

$$\sum_{j=1}^n x_{ij} = 1 \text{ for } i = 1, \dots, n$$

$$\sum_{i=1}^n x_{ij} = 1 \text{ for } j = 1, \dots, n$$

$$x_{ij} \geq 0 \text{ for } i, j = 1, \dots, n.$$

Below we develop an algorithm for the assignment problem. Afterwards we show how the maximum weight bipartite matching problem can be solved as an assignment problem.

Taking the linear programming dual of the assignment problem, we get:

$$w = \min \sum_{i=1}^n u_i + \sum_{j=1}^n v_j$$

$$u_i + v_j \geq c_{ij} \text{ for } i, j = 1, \dots, n.$$

First we make an important observation that is valid for the assignment and traveling salesman problems allowing us to change the cost matrix in a certain way.

Proposition 4.4 For all values $\{u_i\}_{i=1}^n$ and $\{v_j\}_{j=1}^n$, the value of any assignment with weights c_{ij} differs by a constant amount from its value with weights $\bar{c}_{ij} = c_{ij} - u_i - v_j$.

This means that a solution is optimal with weights c_{ij} if and only if it is optimal with weights \bar{c}_{ij} .

Proof. For every primal feasible solution,

$$\begin{aligned} \sum_{i=1}^n \sum_{j=1}^n \bar{c}_{ij} x_{ij} &= \sum_{i=1}^n \sum_{j=1}^n (c_{ij} - u_i - v_j) x_{ij} \\ &= \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} - \sum_{i=1}^n u_i \left(\sum_{j=1}^n x_{ij} \right) - \sum_{j=1}^n v_j \left(\sum_{i=1}^n x_{ij} \right) \\ &= \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} - \sum_{i=1}^n u_i - \sum_{j=1}^n v_j. \end{aligned}$$

So for any feasible solution of the primal, the difference in values of the two objectives are always the constant $\sum_{i=1}^n u_i + \sum_{j=1}^n v_j$, and the claim follows. ■

Now we wish to characterize an optimal solution.

Proposition 4.5 *If there exist values u_i, v_j , and an assignment x such that:*

(i) $\bar{c}_{ij} = c_{ij} - u_i - v_j \leq 0$ for all i, j , and

(ii) $x_{ij} = 1$ only when $\bar{c}_{ij} = 0$,

then the assignment x is optimal and has value $\sum_{i=1}^n u_i + \sum_{j=1}^n v_j$.

Proof. Because $\bar{c}_{ij} \leq 0$ for all i, j , the value of an optimal assignment with weights \bar{c}_{ij} is necessarily nonpositive. But by condition (ii), with weights \bar{c}_{ij} , the assignment x has value $\sum_{i=1}^n \sum_{j=1}^n \bar{c}_{ij} x_{ij} = 0$ and is thus optimal. Now by Proposition 4.4, x is also optimal with weights c_{ij} and has value $\sum_{i=1}^n u_i + \sum_{j=1}^n v_j$. ■

Note that this is another way of writing the linear programming complementarity conditions. In fact (i) tells us that u, v is a dual feasible solution, and (ii) that complementary slackness holds.

Idea of the Algorithm. We are going to use a so-called "primal-dual" algorithm on the graph $G = (V_1, V_2, E)$ where $V_1 = \{1, \dots, n\}$, $V_2 = \{1', \dots, n'\}$ and E consists of all edges with one endpoint in V_1 and the other in V_2 .

At all times we will have a dual feasible solution u, v , or in other words $\bar{c}_{ij} \leq 0$ for all $i \in V_1, j \in V_2$.

Then we will try to find an assignment (a matching of size n) using only the edges $\bar{E} \subseteq E$ where $\bar{E} = \{(i, j) : \bar{c}_{ij} = 0\}$. To do this we will solve the maximum cardinality matching problem on the graph $\bar{G} = (V_1, V_2, \bar{E})$.

If we find a matching of size n , then by Proposition 4.5 we have an optimal weight assignment. Otherwise we return to the dual step and change the dual variables.

Algorithm for the Assignment Problem

Step 0. Let u, v be initial weights such that $\bar{c}_{ij} \leq 0$ for all i, j . Let $\bar{E} = \{(i, j) : \bar{c}_{ij} = 0\}$. Find a maximum cardinality matching M^* in the graph $\bar{G} = (V_1, V_2, \bar{E})$ using the algorithm described in the previous section.

If $|M^*| = n$, stop. M^* is optimal.

Otherwise, note the matching $M = M^*$ and the labeled nodes V_1^+, V_2^+ on termination. Go to Step 2.

Step 1 (Primal Step). Let $\bar{E} = \{(i, j) : \bar{c}_{ij} = 0\}$. M is a feasible matching, and V_1^+, V_2^+ are feasible labels. Continue with the maximum cardinality matching algorithm of the previous section to find an optimal matching M^* .

If $|M^*| = n$, stop. M^* is optimal.

Otherwise, note the matching $M = M^*$ and the labeled nodes V_1^+, V_2^+ on termination. Go to Step 2.

Step 2 (Dual Step). Change the dual variables as follows:

Set $\delta = \min_{i \in V_1^+, j \in V_2 \setminus V_2^+} \{-\bar{c}_{ij}\}$.

Set $u_i \leftarrow u_i - \delta$ for $i \in V_1^+$.

Set $v_j \leftarrow v_j + \delta$ for $j \in V_2^+$.

Return to Step 1. ■

We now need to show that the algorithm terminates correctly, and then see how long it takes.

Proposition 4.6 *Each time that Step 1 terminates with labeled nodes V_1^+, V_2^+ , $|V_1^+| > |V_2^+|$.*

Proof. Every node of V_2^+ touches a matching edge whose other endpoint is in V_1^+ . In addition V_1^+ contains at least one node that is not incident to the matching and received an initial label *. ■

Proposition 4.7 *In the dual step, $\delta > 0$.*

Proof. We observed in the proof of Theorem 4.3 that there are no edges of \bar{E} between V_1^+ and $V_2 \setminus V_2^+$. Therefore $\bar{c}_{ij} < 0$ for all $i \in V_1^+, j \in V_2 \setminus V_2^+$. ■

Proposition 4.8 *After a dual change,*

$\bar{c}_{ij} \leftarrow \bar{c}_{ij}$ for $i \in V_1^+, j \in V_2^+$

$\bar{c}_{ij} \leftarrow \bar{c}_{ij}$ for $i \in V_1 \setminus V_1^+, j \in V_2 \setminus V_2^+$

$\bar{c}_{ij} \leftarrow \bar{c}_{ij} - \delta$ for $i \in V_1 \setminus V_1^+, j \in V_2^+$

$\bar{c}_{ij} \leftarrow \bar{c}_{ij} + \delta$ for $i \in V_1^+, j \in V_2 \setminus V_2^+$

and the new solution is dual feasible.

Proof. \bar{c}_{ij} only increases when $i \in V_1^+, j \in V_2 \setminus V_2^+$. So we must check that the new values are nonpositive. However, δ was chosen precisely so that this condition is satisfied and at least one of these edges now has value $\bar{c}_{ij} = 0$. ■

Proposition 4.9 *The labels V_1^+ and V_2^+ remain valid after a dual change.*

Proof. During a dual change, \bar{c}_{ij} is unchanged for $i \in V_1^+, j \in V_2^+$, and so the labeling remains valid. ■

The above observations tell us that the primal step can restart with the old labels, and the dual objective value decreases by a positive amount $\delta(|V_1^+| - |V_2^+|)$ at each iteration. However, to see that the algorithm runs fast, we can say much more.

Observation 4.2 $|V_2^+|$ increases after a dual change, because of the previous proposition and the choice of δ .

Observation 4.3 The cardinality of the maximum cardinality matching must increase after at most n dual changes, as $|V_2^+|$ cannot exceed n .

Observation 4.4 The cardinality of the maximum cardinality matching can increase at most n times, as $|M^*|$ cannot exceed n .

Proposition 4.10 *The algorithm has complexity $O(n^4)$.*

Proof. By the previous observations, the total number of dual changes in the course of the algorithm is $O(n^2)$. Work in a dual step is $O(|E|)$. The work in the primal step between two augmentations is also $O(|E|)$. ■

Example 4.2 Consider an instance of the assignment problem with $n = 4$ and the profit matrix

$$(c_{ij}) = \begin{pmatrix} 27 & 17 & 7 & 8 \\ 14 & 2 & 10 & 2 \\ 12 & 19 & 4 & 4 \\ 8 & 6 & 12 & 6 \end{pmatrix}.$$

We apply the assignment algorithm. In step 0, we find a first dual feasible solution by setting $v_j^1 = \max_i c_{ij}$ for $j = 1, \dots, n$ and $u_i^1 = 0$ for $i = 1, \dots, n$. This gives a dual feasible solution, and the reduced profit matrix

$$(\bar{c}_{ij}^1) = \begin{pmatrix} 0 & -2 & -5 & 0 \\ -13 & -17 & -2 & -6 \\ -15 & 0 & -8 & -4 \\ -19 & -13 & 0 & -2 \end{pmatrix} \text{ with } u = (0, 0, 0, 0), v = (27, 19, 12, 8).$$

The corresponding dual solution has value $\sum_{i=1}^4 u_i^1 + \sum_{j=1}^4 v_j^1 = 66$. Now we observe that there is no zero entry in the second row of this matrix, so we can immediately improve the dual solution, and more importantly add another

edge to \bar{E} by setting $u_2^2 = \max_j \bar{c}_{2j}^1 = -2$. This gives the new reduced profit matrix

$$(\bar{c}_{ij}^2) = \begin{pmatrix} 0 & -2 & -5 & 0 \\ -11 & -15 & 0 & -4 \\ -15 & 0 & -8 & -4 \\ -19 & -13 & 0 & -2 \end{pmatrix} \text{ with } u = (0, -2, 0, 0), v = (27, 19, 12, 8)$$

and a dual objective value of 64.

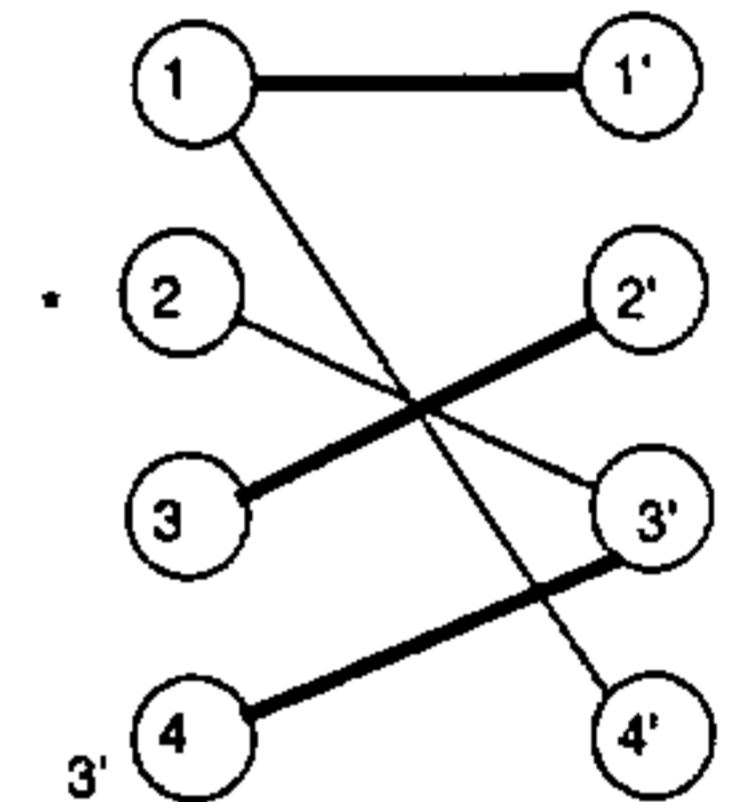


Fig. 4.4 Primal step

In the primal step we now construct the bipartite graph shown in Figure 4.4. We find an initial matching easily by a greedy approach. Suppose we obtain the matching $M = \{(1, 1'), (3, 2'), (4, 3')\}$ with $|M| = 3$. The augmenting path algorithm leads to the labels shown in Figure 4.4.

Node 2 receives the label *.

Node 3' receives the label 2.

Node 4 receives the label 3', and then no more labels can be given.

Thus $V_1^+ = \{2, 4\}$ and $V_2^+ = \{3'\}$.

In the dual step, we find that $\delta = -\bar{c}_{44}^2 = 2$. This leads to a modified dual feasible solution and the reduced profit matrix

$$(\bar{c}_{ij}^3) = \begin{pmatrix} 0 & -2 & -7 & 0 \\ -9 & -13 & 0 & -2 \\ -15 & 0 & -10 & -4 \\ -17 & -11 & 0 & 0 \end{pmatrix} \text{ with } u = (0, -4, 0, -2), v = (27, 19, 14, 8).$$

The dual solution now has value 62.

Now in the primal step the edge $(4, 4')$ is added to \bar{E} . The same labels can be given as before and in addition node 4' now receives the label 4. An augmenting path $\{(2, 3'), (4, 3'), (4, 4')\}$ has been found. $M = \{(1, 1'), (2, 3'), (3, 2'), (4, 4')\}$ is a larger matching, and as it is of size $n = 4$, it is optimal. It can also be checked that its value is 62, equal to that of the dual solution. ■

Now we return to the maximum weight bipartite matching problem. We demonstrate by example how the assignment algorithm can be used to solve it. Consider the instance shown in Figure 4.5.

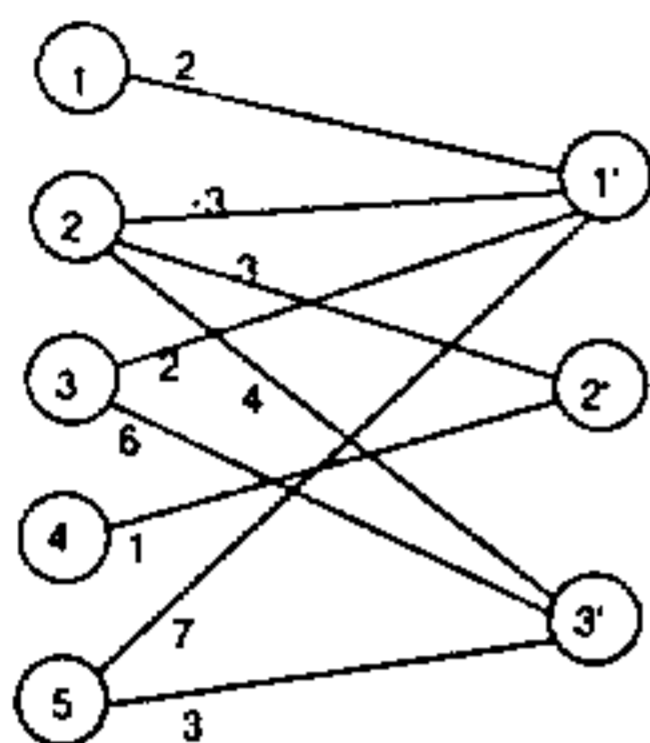


Fig. 4.5 Maximum weight bipartite matching

As $|V_1| - |V_2| = 2$, we add two nodes $4'$ and $5'$ to V_2 . All existing edges are given a weight $\max\{c_e, 0\}$. All missing edges are added with a weight of 0. The resulting assignment weight matrix is

$$(c_{ij}) = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 & 0 \\ 2 & 0 & 6 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 7 & 0 & 3 & 0 & 0 \end{pmatrix}.$$

The assignment algorithm terminates with an optimal solution $x_{15'} = x_{22'} = x_{33'} = x_{44'} = x_{51'} = 1$. The edges with positive weight $(22')$, $(33')$, $(51')$ provide a solution of the matching problem.

4.4 NOTES

More detailed chapters on assignment problems are in [AhuMagOri93], and on general matchings in [CooCunetal97] and [NemWol88]. The complete story on matchings is in the book [LovPlu86].

4.1 The results on alternating paths are from [Ber57] and [NorRab59].

4.3 The primal-dual algorithm can be found in [Kuh55], [ForFul62]. Primal dual algorithms for linear programming were also proposed in [DanForFul56]. Many of the first algorithms for combinatorial optimization problems were primal-dual, such as the matching algorithm [Edm65b] and the matroid intersection algorithm [Edm70]; see also [Law76] and [CooCunetal97].

4.5 EXERCISES

1. Find two augmenting paths for the matching M shown in the graph of Figure 4.6. Is the new matching M' obtained after augmentation optimal? Why?

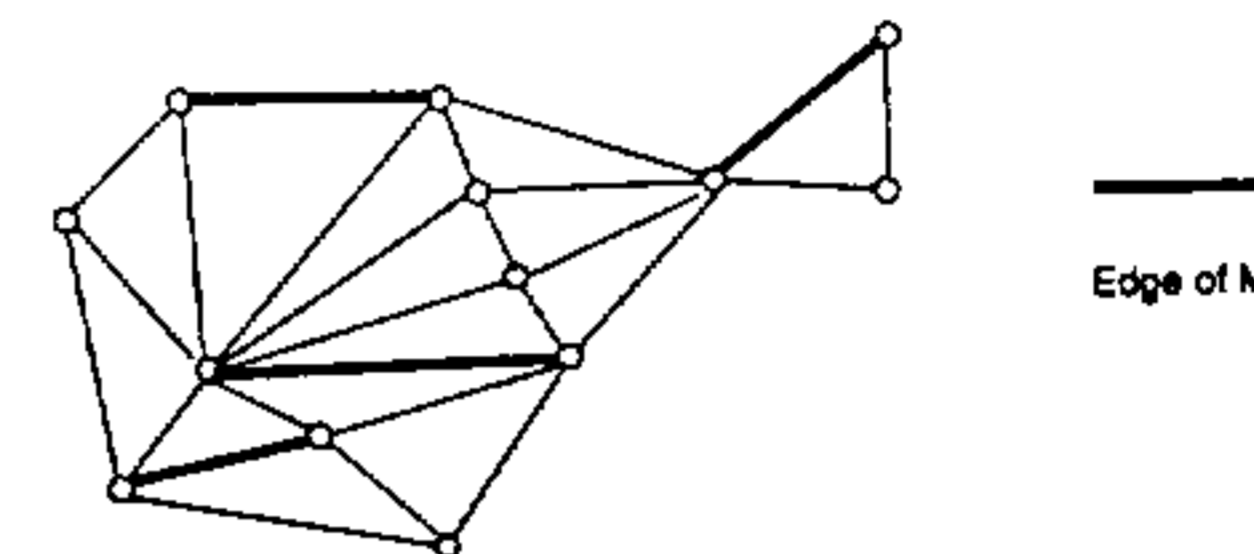


Fig. 4.6 Matching to be augmented

2. Find a maximum cardinality matching in the graph of Figure 4.7.

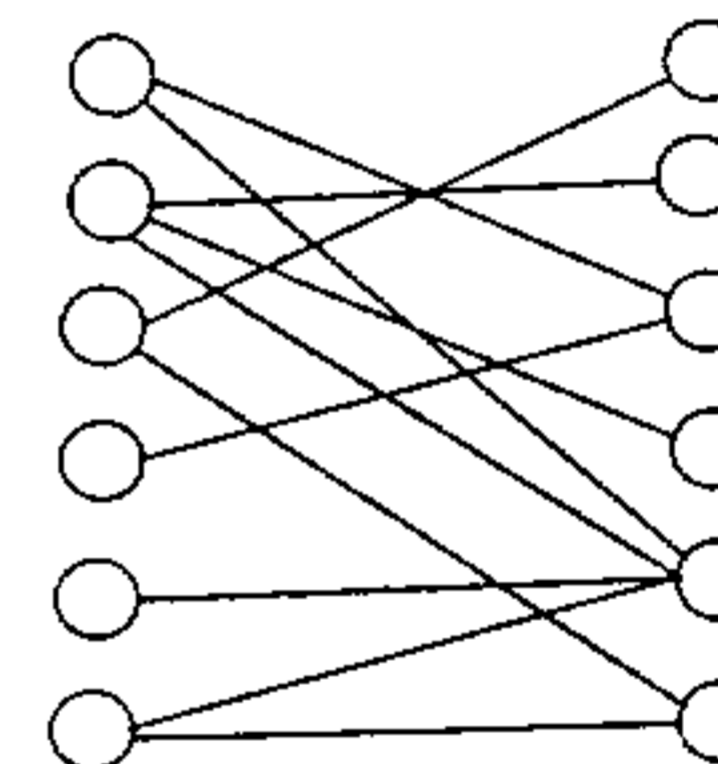


Fig. 4.7 Maximum cardinality matching

Demonstrate that your solution is optimal.

3. If a graph has $n = 2k$ nodes, a matching with k edges is called *perfect*. Show directly that the graph of Figure 4.7 does not contain a perfect matching.

4. Given a connected graph $G = (V, E)$ and positive edge lengths c_e for $e \in E$, the *Chinese Postman* (or *garbage collection*) *Problem* consists of visiting each edge of G at least once beginning and finishing at the same vertex, and minimizing the total distance traveled.

- (i) Show that the minimum distance traveled is $\sum_{e \in E} c_e$ if and only if the graph is *Eulerian* (all nodes have even degree).
- (ii) Show that if G is not Eulerian, and k is the number of nodes of odd degree, then k is even and at least $\frac{k}{2}$ edges must be traversed more than once.
- (iii) Show that the minimum additional distance that must be traveled can be found by solving a minimum weight perfect matching problem in a certain

subgraph (suppose that if $e = (i, j) \in E$, the shortest path between i and j is via edge e).

5. Show how a maximum flow algorithm can be used to find a maximum cardinality matching in a bipartite graph.

6. Find a maximum weight assignment with the weight matrix:

$$(c_{ij}) = \begin{pmatrix} 6 & 2 & 3 & 4 & 1 \\ 9 & 2 & 7 & 6 & 0 \\ 8 & 2 & 1 & 4 & 9 \\ 2 & 1 & 3 & 4 & 4 \\ 1 & 6 & 2 & 9 & 1 \end{pmatrix}.$$

7. Find a maximum weight matching in the weighted bipartite graph of Figure 4.8.

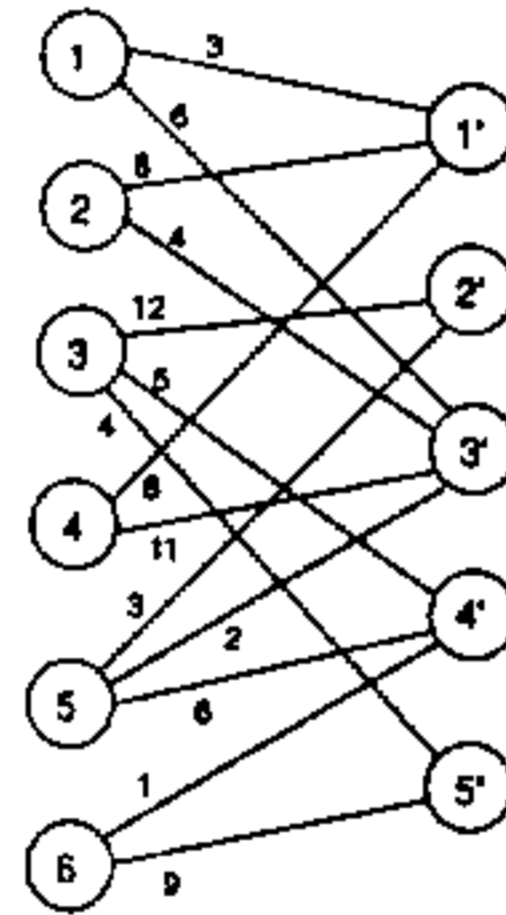


Fig. 4.8 Weighted bipartite graph

8. Show how an algorithm for the maximum weight bipartite matching problem can be used to solve the (equality) assignment problem.

9. Find a lower bound on the optimal value of a 6-city *TSP* instance with distance matrix

$$(c_{ij}) = \begin{pmatrix} - & 2 & 3 & 4 & 1 & 11 \\ 9 & - & 7 & 6 & 0 & 5 \\ 8 & 2 & - & 4 & 9 & 8 \\ 2 & 1 & 3 & - & 4 & 6 \\ 1 & 6 & 2 & 9 & - & 3 \\ 7 & 4 & 6 & 12 & 7 & - \end{pmatrix}.$$

10. Ten researchers are engaged in a set of ten projects. Let S_i denote the researchers working on project i for $i = 1, \dots, 10$. To keep track of progress or problems, management wishes to designate one person working on each project to report at their weekly meeting. Ideally no person should be asked to report on more than one project. Is this possible or not, when $S_1 = \{3, 7, 8, 10\}$, $S_2 = \{4, 8\}$, $S_3 = \{2, 5, 7\}$, $S_4 = \{1, 2, 7, 9\}$, $S_5 = \{2, 5, 7\}$, $S_6 = \{1, 4, 5, 7\}$, $S_7 = \{2, 7\}$, $S_8 = \{1, 6, 7, 10\}$, $S_9 = \{2, 5\}$, $S_{10} = \{1, 2, 3, 6, 7, 8, 10\}$?

11. Suggest an algorithm to solve Exercise 5 of Section 1.9.

9. Formulate the 0-1 knapsack problem as a longest path problem.
10. Derive a dynamic programming recursion for the *STSP* using $f(S, j)$ with $1, j \in S$ where $f(S, j)$ denotes the length of a shortest Hamiltonian path starting at node 1, passing through the nodes of $S \setminus \{1, j\}$ and terminating at node j .
11. Given the weighted rooted subtree of Figure 5.6, devise a dynamic programming algorithm to find optimal weighted subtrees with $k = 2, 3, \dots, n-1$ nodes.
12. Given a tree T and a list of T_1, \dots, T_m of subtrees of T with weights $c(T_i)$ for $i = 1, \dots, m$, describe an algorithm to find a maximum weight packing of node disjoint subtrees.
- 13.* Given a rooted tree T on n nodes, and an n by n matrix C , describe an algorithm to find a maximum weight packing of rooted subtrees of T , where the value of a subtree on node set S with root $i \in S$ is $\sum_{j \in S} c_{ij}$.

6

Complexity and Problem Reductions

6.1 COMPLEXITY

If we consider a list of the problems we have examined so far, we have either shown or it can be shown that the following have the Efficient Optimization Property:

- The Uncapacitated Lot-Sizing Problem (Chapter 5)
- The Maximum Weight Tree Problem (Chapter 3)
- The Maximum Weight Matching Problem
- The Shortest Path Problem (Chapter 5)
- The Max Flow Problem (Chapter 3)
- The TU Integer Programming Problem (Chapter 3)
- The Assignment Problem (Chapter 4)

Below we make this more precise: there is a polynomial algorithm for these optimization problems.

On the other hand, no one to date has found an efficient (polynomial) algorithm for any of the following optimization problems:

- The 0-1 Knapsack Problem (Chapter 1)
- The Set Covering Problem (Chapter 1)
- The Traveling Salesman Problem (Chapter 1)
- The Uncapacitated Facility Location Problem (Chapter 1)
- The Integer Programming Problem (Chapter 1)
- The Steiner Tree Problem (Chapter 3)

The remainder of this book will in large part be devoted to examining how to tackle problems in this second group. However, it is first useful to discuss the distinction (real or imaginary) between these two groups, so that when we encounter a new optimization problem we have an idea how we might classify and then attempt to solve it.

To develop such a method of classification, we need just four concepts:

- A class C of legitimate problems to which the theory applies
- A nonempty subclass $C_A \subseteq C$ of "easy" problems
- A nonempty subclass $C_B \subseteq C$ of "difficult" problems
- A relation "not more difficult than" between pairs of legitimate problems.

This immediately leads to:

Proposition 6.1 (Reduction Lemma) *Suppose that P and Q are two legitimate problems.*

- If Q is "easy" and P is "not more difficult than" Q , then P is "easy".*
- If P is "difficult" and P is "not more difficult than" Q , then Q is "difficult".*

We have already used the first part of the lemma implicitly in Chapter 4. There we show that the maximum weight bipartite matching problem is "easy" by showing that it is reducible to the assignment problem. Also Exercise 4.4 involves showing that the Chinese postman problem is reducible to maximum weight matching. The goal of the rest of this chapter is to somewhat formalize these notions. In the next section we introduce the class of legitimate problems and the "easy" class, and in Section 6.3 we discuss the concept of problem reduction which allows us to then define the "difficult" class. By the end of the chapter we will then have another tool at our disposal: namely a way to show that certain problems are "difficult" by using the second part of the reduction lemma.

6.2 DECISION PROBLEMS, AND CLASSES \mathcal{NP} AND \mathcal{P}

Unfortunately, the theory does not exactly address optimization problems in the form we have posed them so far. To define the class of legitimate problems, it is appropriate to pose decision problems having YES-NO answers. Thus an optimization problem:

$$\max\{cx : x \in S\}$$

for which an instance consists of: $\{c$ and a "standard" representation of $S\}$ is replaced by the *decision problem*:

Is there an $x \in S$ with value $cx \geq k$?

for which an instance consists of $\{c$, a "standard" representation of S , and an integer $k\}$.

So for the rest of this chapter (unless explicitly stated), when we refer to an optimization problem TSP , UFL , and so on, we have in mind the corresponding decision problem.

Next we give a slightly more formal definition of the running time of an algorithm than that given at the start of Chapter 3. It is not just the number of variables and constraints or nodes and edges that defines the input length, but also the size of the numbers occurring in the data.

Definition 6.1 For a problem instance X , the *length of the input* $L = L(X)$ is the length of the binary representation of a "standard" representation of the instance.

Definition 6.2 Given a problem P , an algorithm A for the problem, and an instance X , let $f_A(X)$ be the number of elementary calculations required to run the algorithm A on the instance X . $f_A^*(l) = \sup_X \{f_A(X) : L(X) = l\}$ is the *running time* of algorithm A . An algorithm A is *polynomial* for a problem P if $f_A^*(l) = O(l^p)$ for some positive integer p .

Now we can define the class of "legitimate" problems.

Definition 6.3 \mathcal{NP} is the class of decision problems with the property that: for any instance for which the answer is YES, there is a "short" (polynomial) proof of the YES.

We note immediately that if a decision problem associated to an optimization problem is in \mathcal{NP} , then the optimization problem can be solved by answering the decision problem a polynomial number of times (by using bisection on the objective function value).

Proposition 6.2 *For each optimization problem in the two lists in Section 6.1, the associated decision problem: "Does there exist a primal solution of value as good as or better than k ?" lies in \mathcal{NP} .*

Now we can define the class of "easy" problems.

Definition 6.4 \mathcal{P} is the class of decision problems in \mathcal{NP} for which there exists a polynomial algorithm.

Example 6.1 (i) Uncapacitated Lot Sizing. Consider ULS for which a dynamic programming algorithm is presented in Chapter 5. For an instance X with integral data (n, d, p, h, f, k) , the input has length $L(X) = \sum_{j=1}^n \lceil \log d_j \rceil + \sum_{j=1}^n \lceil \log p_j \rceil + \sum_{j=1}^n \lceil \log h_j \rceil + \sum_{j=1}^n \lceil \log f_j \rceil + \lceil \log k \rceil$.

The DP algorithm requires only $O(n^2)$ additions and comparisons of the numbers occurring in the data, and hence the size of numbers required to give a YES answer, and the running time are certainly $O(L^2)$. Thus ULS is in \mathcal{P} .

(ii) 0-1 Knapsack. For an instance X of 0-1 KNAPSACK: $\{\sum_{j=1}^n c_j x_j \geq k, \sum_{j=1}^n a_j x_j \leq b, x \in \{0, 1\}^n\}$, the length of the input is $L(X) = \sum_{j=1}^n \lceil \log c_j \rceil + \sum_{j=1}^n \lceil \log a_j \rceil + \lceil \log b \rceil + \lceil \log k \rceil$.

For an instance for which the answer is YES, it suffices to (a) read a solution $x^* \in \{0, 1\}^n$, and (b) check that $ax^* \leq b$ and $cx^* \geq k$. Both (a) and (b) can be carried out in time polynomial in L , so the associated decision problem is in \mathcal{NP} .

From Section 5.4, dynamic programming provides an $O(nb)$ algorithm. As b is not equal to $(\log b)^p$ for any fixed p , this algorithm is not polynomial, and in fact no polynomial algorithm is known for 0-1 KNAPSACK.

(iii) Symmetric Traveling Salesman. For STSP with instance $(G, c^{|E|}, k)$, it suffices to check that a proposed set of edges forms a tour and that its length does not exceed k . The argument for most other problems on the second list is similar.

(iv) Integer Programming. Problem IP requires a little more work, because one needs to show that there always exists an optimal solution x^* whose description length $\sum_{j=1}^n \lceil \log x_j^* \rceil$ is polynomial in L . ■

Do the second set of optimization problems listed in Section 6.1 above have anything in common apart from the fact that their decision problems are in \mathcal{NP} , and that nobody has yet discovered a polynomial algorithm for any of them?

Surprisingly, they have a second property in common: their decision problems are all among the *most difficult problems* in \mathcal{NP} .

6.3 POLYNOMIAL REDUCTION AND THE CLASS \mathcal{NPC}

This is the formal definition of "is not more difficult than" that we need.

Definition 6.5 If $P, Q \in \mathcal{NP}$, and if an instance of P can be converted in polynomial time to an instance of Q , then P is *polynomially reducible* to Q .

Note that this means that if we have an algorithm for problem Q , it can be used to solve problem P with an overhead that is polynomial in the size of the instance. We now define the class of "most difficult" problems.

Definition 6.6 \mathcal{NPC} , the class of \mathcal{NP} -complete problems, is the subset of problems $P \in \mathcal{NP}$ such that for all $Q \in \mathcal{NP}$, Q is polynomially reducible to P .

It is a remarkable fact not only that \mathcal{NPC} is nonempty, but that all of the decision problems in our second list are in \mathcal{NPC} . So how can one prove that a problem is in \mathcal{NPC} ?

The most important step is to prove that \mathcal{NPC} is nonempty. Written as an 0-1 integer program, SATISFIABILITY is the decision problem:

Given $N = \{1, \dots, n\}$, and $2m$ subsets $\{C_i\}_{i=1}^m$ and $\{D_i\}_{i=1}^m$ of N , does the 0-1 integer program:

$$\sum_{j \in C_i} x_j + \sum_{j \in D_i} (1 - x_j) \geq 1 \text{ for } i = 1, \dots, m$$

$$x \in B^n$$

have a feasible solution?

It is obvious that this problem is in \mathcal{NP} . Cook showed in 1970 that SATISFIABILITY is in \mathcal{NPC} .

Now we indicate how the reduction lemma can be used to show that all the problems of the second list and many others are in \mathcal{NPC} .

For example, to see that BIP is in \mathcal{NPC} , all we need to observe is that

(i) $BIP \in \mathcal{NP}$, (which is immediate) and

(ii) SATISFIABILITY reduces to BIP. (Above we actually described SATISFIABILITY as a BIP, and so this is also immediate).

We now restate the Reduction Lemma (Proposition 6.1) more formally.

Proposition 6.3 Suppose that problems $P, Q \in \mathcal{NP}$.

(i) If $Q \in \mathcal{P}$ and P is polynomially reducible to Q , then $P \in \mathcal{P}$.

(ii) If $P \in \mathcal{NPC}$ and P is polynomially reducible to Q , then $Q \in \mathcal{NPC}$.

Proof. (ii) Consider any problem $R \in \mathcal{NP}$. As $P \in \mathcal{NPC}$, R is polynomially reducible to P . However P is polynomially reducible to Q by hypothesis, and thus R is polynomially reducible to Q . As this holds for all $R \in \mathcal{NP}$, $Q \in \mathcal{NPC}$. ■

This has an important corollary.

Corollary 6.1 If $\mathcal{P} \cap \mathcal{NPC} \neq \emptyset$, then $\mathcal{P} = \mathcal{NP}$.

Proof. Suppose $Q \in \mathcal{P} \cap \mathcal{NPC}$ and take $R \in \mathcal{NP}$. As $R \in \mathcal{NP}$ and $Q \in \mathcal{NPC}$, R is polynomially reducible to Q . By (i), as $Q \in \mathcal{P}$ and R is polynomially reducible to Q , $R \in \mathcal{P}$. So $\mathcal{NP} \subseteq \mathcal{P}$ and thus $\mathcal{P} = \mathcal{NP}$. ■

The list of problems known to be in \mathcal{NPC} is now enormous. Some of the most basic problems in \mathcal{NPC} are the problems in our second list. Below we prove that the *Capacitated Lot-Sizing Problem (CLS)* can be added to the list.

Example 6.2 Consider the lot-sizing problem introduced in Chapter 1, with a production capacity constraint in each period. *CLS* has a formulation:

$$\begin{aligned} \min \quad & \sum_{t=1}^n p_t x_t + \sum_{t=1}^n h_t s_t + \sum_{t=1}^n f_t y_t \\ & s_{t-1} + x_t = d_t + s_t \text{ for } t = 1, \dots, n \\ & x_t \leq C_t y_t \text{ for } t = 1, \dots, n \\ & s_0 = s_n = 0, s \in R_+^{n+1}, x \in R_+^n, y \in B^n. \end{aligned}$$

First, is the decision version of *CLS* in *NP*? One answer lies in the observation that there is always an optimal solution of the form: $y \in \{0, 1\}^n$ with x a basic feasible solution of the network flow problem in which y is fixed. So there exists an optimal solution whose length is polynomial in the length of the input, and can be used to verify a YES answer in polynomial time. One just needs to check that it satisfies the constraints and its value is sufficiently small.

We now show that 0-1 *KNAPSACK* is polynomially reducible to *CLS*. To do this, we show how an instance of the 0-1 knapsack problem can be solved as a capacitated lot-sizing problem. Given an instance:

$$\min \left\{ \sum_{j=1}^n c_j y_j : \sum_{j=1}^n a_j y_j \geq b, y \in B^n \right\},$$

we solve a lot-sizing instance with n periods, $p_t = h_t = 0, f_t = c_t, C_t = a_t$ for all $t, d_t = 0$ for $t = 1, \dots, n-1$ and $d_n = b$.

An equivalent formulation of the lot-sizing problem, obtained by eliminating the stock variables as described in Section 1.4, is:

$$\begin{aligned} \min \quad & \sum_{t=1}^n p'_t x_t + \sum_{t=1}^n f_t y_t \\ & \sum_{i=1}^t x_i \geq \sum_{i=1}^t d_i \text{ for } t = 1, \dots, n-1 \\ & \sum_{i=1}^n x_i = \sum_{i=1}^n d_i \\ & x_t \leq C_t y_t \text{ for } t = 1, \dots, n \\ & x \in R_+^n, y \in B^n. \end{aligned}$$

Rewriting this with the chosen values for the data, we obtain:

$$\begin{aligned} \min \quad & \sum_{t=1}^n c_t y_t \\ & \sum_{i=1}^t x_i \geq 0 \text{ for } t = 1, \dots, n-1 \end{aligned}$$

$$\begin{aligned} \sum_{i=1}^n x_i &= b \\ x_t &\leq a_t y_t \text{ for } t = 1, \dots, n \\ x &\in R_+^n, y \in B^n. \end{aligned}$$

Dropping the $n-1$ redundant demand constraints leaves

$$\begin{aligned} \min \quad & \sum_{t=1}^n c_t y_t \\ & \sum_{t=1}^n x_t = b \\ & x_t \leq a_t y_t \text{ for } t = 1, \dots, n \\ & x \in R_+^n, y \in B^n. \end{aligned}$$

Now let (x^*, y^*) be an optimal solution of this lot-sizing instance. Combining the constraint $\sum_{t=1}^n x_t = b$ with the constraints $x_t \leq a_t y_t$ for $t = 1, \dots, n$, we see that $\sum_t a_t y_t^* \geq b$ and so y^* is feasible in the knapsack instance. It is also optimal, because a better knapsack solution \bar{y} with $c\bar{y} < cy^*$ would also provide a better lot-sizing solution $((\bar{x})_t = a_t \bar{y}_t, \bar{y})$. So an optimal y vector for the lot-sizing instance solves the knapsack instance. So 0-1 *KNAPSACK* is polynomially reducible to *CLS*, and as 0-1 *KNAPSACK* $\in NPC$, *CLS* $\in NPC$. ■

6.4 CONSEQUENCES OF $P = NP$ OR $P \neq NP$

Most problems of interest have either been shown to be in P or in *NPC*. What is more, nobody has succeeded either in proving that $P = NP$ or in showing that $P \neq NP$. However, given the huge number of problems in *NPC* for which no polynomial algorithm has been found, it is a practical working hypothesis that $P \neq NP$.

So how should we interpret the above results and observations?

A first important remark concerns the class *NP*. Typically, problems in this class have a very large (exponentially large) set of feasible solutions, and these problems can in theory be solved by enumerating the feasible solutions. As we saw in Table 1.1, this is impractical for instances of any reasonable size.

A pessimist might say that as most problems appear to be hard (i.e., their decision version lies in *NPC*), we have no hope of solving instances of large size (because in the worst case we cannot hope to do better than enumeration), and so we should give up.

A mathematician (optimist) might set out to become famous by proving that $P = NP$.

A mathematician (pessimist) might set out to become famous by proving that $P \neq NP$.

A mathematician (*thoughtful*) might decide to ask a different question: Can I find an algorithm that is guaranteed to find a solution "close to optimal" in polynomial time in all cases?

A probabilist (*thoughtful*) might also ask a different question: Can I find an algorithm that runs in polynomial time with high probability and that is guaranteed to find an optimal or "close to optimal" solution with high probability?

An engineer would start looking for a heuristic algorithm that produces practically usable solutions.

Your boss might say: I don't care a damn about integer programming theory. You just worry about our scheduling problem. Give me a feasible production schedule for tomorrow in which William Brown and Daughters' order is out of the door by 4 P.M.

A struggling professor might say: Great. Previously I was trying to develop one algorithm to solve all integer programs, and publishing one paper every two years explaining why I was not succeeding. Now I know that I might as well study each \mathcal{NP} problem individually. As there are thousands of them, I should be able to write twenty papers a year.

Needless to say they are nearly all right. There is no easy and rapid solution, but the problems will not go away, and more and more fascinating and important practical problems are being formulated as integer programs. So in spite of the \mathcal{NP} -completeness theory, using an appropriate combination of theory, algorithms, experience, and intensive calculation, verifiably good solutions for large instances can and must be found.

Definition 6.7 An optimization problem for which the decision problem lies in \mathcal{NPC} is called \mathcal{NP} -hard.

The following chapters are devoted to ways to tackle such \mathcal{NP} -hard problems. First, however, we return briefly to the *Separation Problem* introduced in Chapter 3.

6.5 OPTIMIZATION AND SEPARATION

Here we consider the question of whether there are ties between problems in \mathcal{P} . How can we show that a problem is in \mathcal{P} ? The most obvious way is by finding a polynomial algorithm. We have also seen that another indirect way is by reduction.

There is, however, one general and important result tying together pairs of problems. Put imprecisely, it says:

Given a family of polyhedra associated with a class of problems (such as the convex hulls of the incidence vectors of feasible points $S \subseteq B^n$), the family of

optimization problems:

$$\max\{cx : x \in \text{conv}(S)\}$$

is polynomially solvable if and only if the family of separation problems:

Is $x \in \text{conv}(S)$? If not, find an inequality satisfied by all points of S , but cutting off x .

is polynomially solvable.

In other words, the Efficient Optimization and Efficient Separation Properties introduced in Chapter 3 are really equivalent. The other two properties are not exactly equivalent. As we indicated earlier, if a problem has the Efficient Separation Property, it suggests that it may have the Explicit Convex Hull Property. Also if a problem has the Explicit Convex Hull Property, then its linear programming dual may lead to the Strong Dual Property.

6.6 NOTES

6.2 An important step in the development of the distinction between easy and difficult problems is the concept of a certificate of optimality [Edm65a], [Edm65b].

6.3 Cook [Coo71] formally introduced the class \mathcal{NP} and showed the existence of an \mathcal{NP} -complete problem. The reduction of many decision versions of integer and combinatorial optimization problems to a \mathcal{NP} -complete problem was shown in [Karp72], [Karp75]. The book [GarJoh79] lists an enormous number of \mathcal{NP} -complete problems and their reductions. A recent update is [CreKan95].

6.5 The equivalence of optimization and separation is shown in [GroLovSch81], [GroLovSch84]. A more thorough exploration of the equivalence appears in [GroLovSch88]. Other results of importance for integer programming concern the difficulty of finding a short description of all facets for \mathcal{NP} -hard problems [PapYan84], and the polynomiality of integer programming with a fixed number of variables [Len83]. Some separation problems are examined in Chapter 9.

For a general book on computational complexity, see [Pap94].

6.7 EXERCISES

1. The 2-PARTITION problem is specified by n positive integers (a_1, \dots, a_n) . The problem is to find a subset $S \subset N = \{1, \dots, n\}$ such that $\sum_{j \in S} a_j =$

8

Cutting Plane Algorithms

8.1 INTRODUCTION

Here we consider the general integer program:

$$(IP) \quad \max\{cx : x \in X\}$$

where $X = \{x : Ax \leq b, x \in Z_+^n\}$.

Proposition 8.1 *$\text{conv}(X) = \{x : \tilde{A}x \leq \tilde{b}, x \geq 0\}$ is a polyhedron.*

This result, already presented in Chapter 1, tells us that we can, in theory, reformulate problem IP as the linear program:

$$(LP) \quad \max\{cx : \tilde{A}x \leq \tilde{b}, x \geq 0\}$$

and then for any value of c , an optimal extreme point solution of LP is an optimal solution of IP . The same result holds for mixed integer programs with $X = \{(x, y) \in R_+^n \times Z_+^p : Ax + Gy \leq b\}$ provided the data A, G, b are rational.

In Chapter 3 we have seen several problems, including the assignment problem and the spanning tree problem, for which we have given an explicit description of $\text{conv}(X)$. However, unfortunately for \mathcal{NP} -hard problems, there is almost no hope of finding a "good" description. Given an instance of an \mathcal{NP} -hard problem, the goal in this chapter is to find effective ways to try to approximate $\text{conv}(X)$ for the given instance.

The fundamental concept that we have already used informally is that of a valid inequality.

Definition 8.1 An inequality $\pi x \leq \pi_0$ is a *valid inequality* for $X \subseteq R^n$ if $\pi x \leq \pi_0$ for all $x \in X$.

If $X = \{x \in Z^n : Ax \leq b\}$ and $\text{conv}(X) = \{x \in R^n : \bar{A}x \leq \bar{b}\}$, the constraints $a^i x \leq b_i$ and $\bar{a}^i x \leq \bar{b}_i$ are clearly valid inequalities for X .

The two questions that immediately spring to mind are

- (i) Which are the "good" or useful valid inequalities? and
- (ii) If we know a set or family of valid inequalities for a problem, how can we use them in trying to solve a particular instance?

8.2 SOME SIMPLE VALID INEQUALITIES

First we present some examples of valid inequalities. The first type, logical inequalities, have already been seen in Example 7.5 in looking at preprocessing.

Example 8.1 A Pure 0-1 Set. Consider the 0-1 knapsack set:

$$X = \{x \in B^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2\}.$$

If $x_2 = x_4 = 0$, the lhs (left-hand side) = $3x_1 + 2x_3 + x_5 \geq 0$ and the rhs (right-hand side) = -2 , which is impossible. So all feasible solutions satisfy the valid inequality $x_2 + x_4 \geq 1$.

If $x_1 = 1$ and $x_2 = 0$, the lhs = $3 + 2x_3 - 3x_4 + x_5 \geq 3 - 3 = 0$ and the rhs = -2 , which is impossible, so $x_1 \leq x_2$ is also a valid inequality. ■

Example 8.2 A Mixed 0-1 Set. Consider the set:

$$X = \{(x, y) : x \leq 9999y, 0 \leq x \leq 5, y \in B^1\}.$$

It is easily checked that the inequality

$$x \leq 5y$$

is valid because $X = \{(0, 0), (x, 1) \text{ with } 0 \leq x \leq 5\}$. As X only involves two variables, it is possible to represent X graphically, so it is easy to check that the addition of the inequality $x \leq 5y$ gives us the convex hull of X .

Such constraints arise often. For instance, in the capacitated facility location problem one has the feasible region:

$$\begin{aligned} \sum_{i \in M} x_{ij} &\leq b_j y_j \text{ for } j \in N \\ \sum_{j \in N} x_{ij} &= a_i \text{ for } i \in M \\ x_{ij} &\geq 0 \text{ for } i \in M, j \in N, y_j \in \{0, 1\} \text{ for } j \in N. \end{aligned}$$

All feasible solutions satisfy $x_{ij} \leq b_j y_j$ and $x_{ij} \leq a_i$ with $y_j \in B^1$. This is precisely the situation above leading to the family of valid inequalities $x_{ij} \leq \min\{a_i, b_j\} y_j$. ■

Example 8.3 A Mixed Integer Set. Consider the set

$$X = \{(x, y) : x \leq 10y, 0 \leq x \leq 14, y \in Z_+^1\}.$$

It is not difficult to verify the validity of the inequality $x \leq 6 + 4y$, or written another way, $x \leq 14 - 4(2 - y)$. In Figure 8.1 we represent X graphically, and see that the addition of the inequality $x \leq 6 + 4y$ gives the convex hull of X .

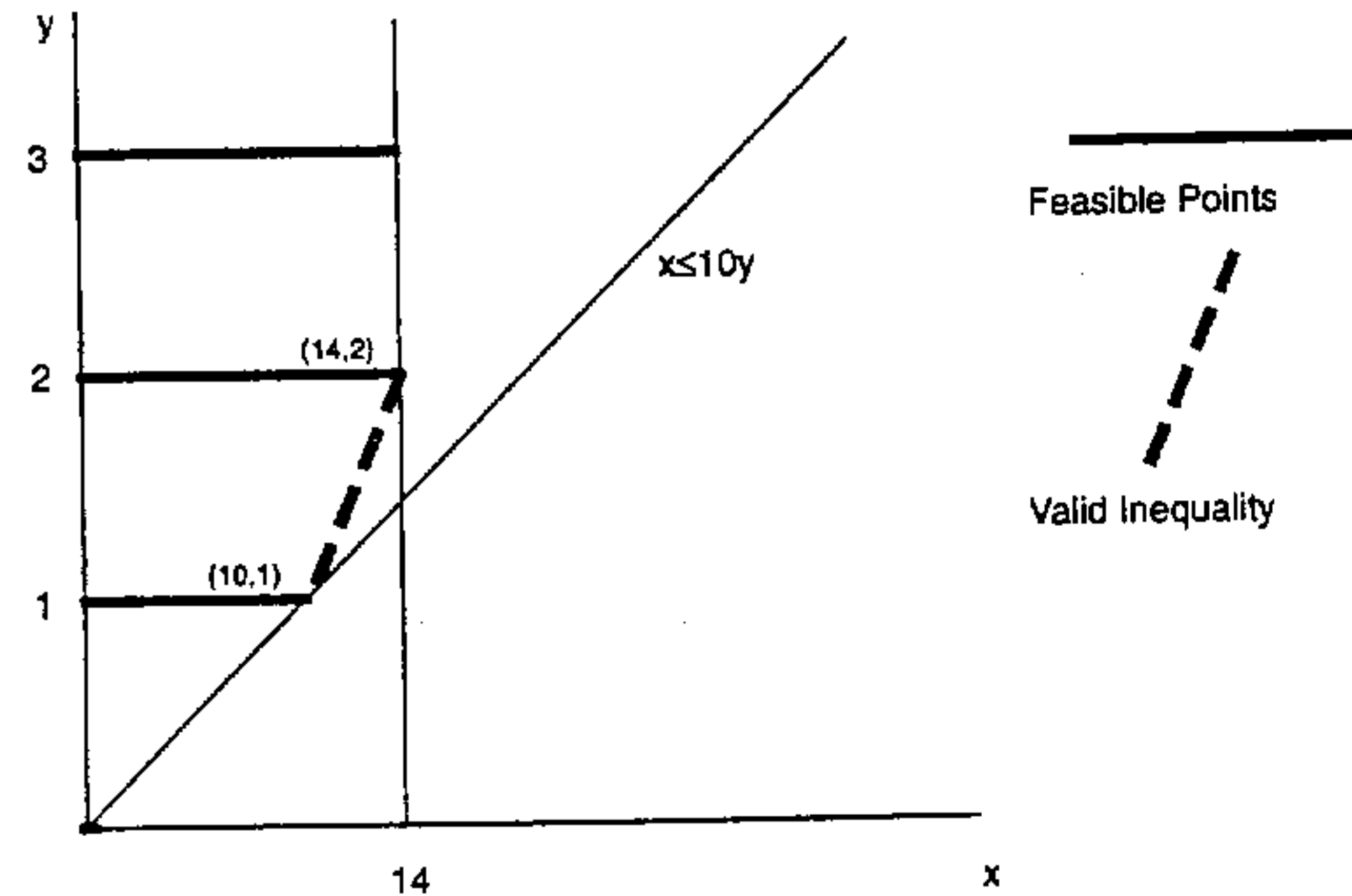


Fig. 8.1 Mixed integer inequality

For the general case, when C does not divide b , and

$$X = \{(x, y) : x \leq Cy, 0 \leq x \leq b, y \in Z_+^1\},$$

one obtains the valid inequality $x \leq b - \gamma(K - y)$ where $K = \lceil \frac{b}{C} \rceil$ and $\gamma = b - (\lceil \frac{b}{C} \rceil - 1)C$. ■

Example 8.4 A Combinatorial Set: Matchings. Consider the X of incidence vectors of matchings:

$$\sum_{e \in \delta(i)} x_e \leq 1 \text{ for } i \in V \tag{8.1}$$

$$x \in Z_+^{|E|} \tag{8.2}$$

where $\delta(i) = \{e \in E : e = (i, j) \text{ for some } j \in V\}$.

Take a set $T \subseteq V$ of nodes of odd cardinality. As the edges of a matching are disjoint, the number of edges of a matching having both endpoints in T is at most $\frac{|T|-1}{2}$. Therefore

$$\sum_{e \in E(T)} x_e \leq \frac{|T|-1}{2} \quad (8.3)$$

is a valid inequality for X if $|T| \geq 3$ and $|T|$ is odd. ■

Example 8.5 Integer Rounding. Consider the integer region $X = P \cap Z^4$ where

$$P = \{x \in R_+^4 : 13x_1 + 20x_2 + 11x_3 + 6x_4 \geq 72\}.$$

Dividing by 11 gives the valid inequality for P :

$$\frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq 6\frac{6}{11}.$$

As $x \geq 0$, rounding up the coefficients on the left to the nearest integer gives $2x_1 + 2x_2 + x_3 + x_4 \geq \frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq 6\frac{6}{11}$, and so we get a weaker valid inequality for P :

$$2x_1 + 2x_2 + x_3 + x_4 \geq 6\frac{6}{11}.$$

As x is integer and all the coefficients are integer, the lhs must be integer. An integer that is greater than or equal to $6\frac{6}{11}$ must be at least 7, and so we can round the rhs up to the nearest integer giving the valid inequality for X :

$$2x_1 + 2x_2 + x_3 + x_4 \geq 7. \quad \blacksquare$$

Such regions arise in many problems. Consider, for instance, a *Generalized Transportation Problem* where the problem is to satisfy the demand d_j of client j using trucks of different types. A truck of type i has capacity C_i , there are a_i of them available, and the cost if a truck of type i is sent to client j is c_{ij} . The resulting integer program is:

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{i=1}^n C_i x_{ij} \geq d_j \text{ for } j = 1, \dots, n \\ & \sum_{j=1}^n x_{ij} \leq a_i \text{ for } i = 1, \dots, m \\ & x \in Z_+^{mn}, \end{aligned}$$

where each demand constraint gives rise to a set of the form X .

Example 8.6 Mixed Integer Rounding. Consider the same example as above with the addition of a continuous variable. Let $X = P \cap (Z^4 \times R^1)$ where

$$P = \{(y, s) \in R_+^4 \times R_+^1 : 13y_1 + 20y_2 + 11y_3 + 6y_4 + s \geq 72\}.$$

In terms of the generalized transportation model, there are four types of truck available to satisfy demand, but it is also possible to satisfy demand from an alternative source. Dividing by 11 gives

$$\frac{13}{11}y_1 + \frac{20}{11}y_2 + y_3 + \frac{6}{11}y_4 \geq \frac{72-s}{11},$$

suggesting that there is a valid inequality

$$2y_1 + 2y_2 + y_3 + y_4 + \alpha s \geq 7 \text{ for some } \alpha. \quad (8.4)$$

Looking at the rhs term $\frac{72-s}{11}$, we see that the rhs $\lceil \frac{72-s}{11} \rceil$ decreases from 7 to 6 at the critical value $s = 6$, indicating the value $\alpha = \frac{1}{6}$. Inequality (8.4) turns out to be valid for values of $\alpha \geq \frac{1}{6}$, and later we will see that it can even be strengthened a little, giving:

$$\frac{3}{2}y_1 + 2y_2 + y_3 + y_4 + \frac{1}{6}s \geq 7. \quad \blacksquare$$

8.3 VALID INEQUALITIES

To understand how to generate valid inequalities for integer programs, it is first necessary to understand valid inequalities for polyhedra (or linear programs).

8.3.1 Valid Inequalities for Linear Programs

So the first question is: When is the inequality $\pi x \leq \pi_0$ valid for $P = \{x : Ax \leq b, x \geq 0\}$?

Proposition 8.2 $\pi x \leq \pi_0$ is valid for $P = \{x : Ax \leq b, x \geq 0\} \neq \emptyset$ if and only if

there exist $u \geq 0, v \geq 0$ such that $uA - v = \pi$ and $ub \leq \pi_0$, or alternatively there exists $u \geq 0$ such that $uA \geq \pi$ and $ub \leq \pi_0$.

Proof. By linear programming duality, $\max\{\pi x : x \in P\} \leq \pi_0$ if and only if $\min\{ub : uA - v = \pi, u \geq 0, v \geq 0\} \leq \pi_0$. ■

8.3.2 Valid Inequalities for Integer Programs

Now we consider the feasible region of an integer program:

$$\{x : Ax \leq b, x \in Z^n\}$$

and ask the same question.

Surprisingly, the complete answer is in some sense given in the following very simple observation.

Proposition 8.3 *Let $X = \{y \in Z^1 : y \leq b\}$, then the inequality $y \leq \lfloor b \rfloor$ is valid for X .*

We have already used this idea in Example 8.5. We now give two more examples.

Example 8.4 (cont) Valid Inequalities for Matching. Here we give an alternative algebraic justification for the validity of inequality (8.3) that can be broken up into three steps.

(i) Take a nonnegative linear combination of the constraints (8.1) with weights $u_i = \frac{1}{2}$ for $i \in T$ and $u_i = 0$ for $i \in V \setminus T$. This gives the valid inequality:

$$\sum_{e \in E(T)} x_e + \frac{1}{2} \sum_{e \in \delta(T, V \setminus T)} x_e \leq \frac{|T|}{2}.$$

(ii) Because $x_e \geq 0$, $\sum_{e \in E(T)} x_e \leq \sum_{e \in E(T)} x_e + \frac{1}{2} \sum_{e \in \delta(T, V \setminus T)} x_e$, and so

$$\sum_{e \in E(T)} x_e \leq \frac{|T|}{2}$$

is a valid inequality.

(iii) Because $x \in Z^n$, the lhs $\sum_{e \in E(T)} x_e$ must be an integer, and so one can replace the rhs value by the largest integer less than or equal to the rhs value. So

$$\sum_{e \in E(T)} x_e \leq \lfloor \frac{|T|}{2} \rfloor$$

is a valid inequality. ■

Example 8.7 Valid Inequalities for an Integer Program. An identical approach can be used to derive valid inequalities for any integer programming region. Let $X = P \cap Z^n$ be the set of integer points in P where P is given by:

$$\begin{aligned} 7x_1 - 2x_2 &\leq 14 \\ x_2 &\leq 3 \\ 2x_1 - 2x_2 &\leq 3 \\ x &\geq 0. \end{aligned}$$

(i) First combining the constraints with nonnegative weights $u = (\frac{2}{7}, \frac{37}{63}, 0)$, we obtain the valid inequality for P

$$2x_1 + \frac{1}{63}x_2 \leq \frac{121}{21}.$$

(ii) Reducing the coefficients on the left-hand side to the nearest integer gives the valid inequality for P :

$$2x_1 + 0x_2 \leq \frac{121}{21}.$$

(iii) Now as the left-hand side is integral for all points of X , we can reduce the rhs to the nearest integer, and we obtain the valid inequality for X :

$$2x_1 \leq \lfloor \frac{121}{21} \rfloor = 5.$$

Observe that if we repeat the procedure, and use a weight of $\frac{1}{2}$ on this last constraint, we obtain the tighter inequality $x_1 \leq \lfloor \frac{5}{2} \rfloor = 2$. ■

Now it is easy to describe the general procedure that we have been using.

Chvátal-Gomory procedure to construct a valid inequality for the set $X = P \cap Z^n$, where $P = \{x \in R_+^n : Ax \leq b\}$, A is an $m \times n$ matrix with columns $\{a_1, a_2, \dots, a_n\}$, and $u \in R_+^m$:

(i) the inequality

$$\sum_{j=1}^n u a_j x_j \leq u b$$

is valid for P as $u \geq 0$ and $\sum_{j=1}^n a_j x_j \leq b$,

(ii) the inequality

$$\sum_{j=1}^n \lfloor u a_j \rfloor x_j \leq u b$$

is valid for P as $x \geq 0$,

(iii) the inequality

$$\sum_{j=1}^n \lfloor u a_j \rfloor x_j \leq \lfloor u b \rfloor$$

is valid for X as x is integer, and thus $\sum_{j=1}^n \lfloor u a_j \rfloor x_j$ is integer.

The surprising fact is that this simple procedure is sufficient to generate all valid inequalities for an integer program.

Theorem 8.4 Every valid inequality for X can be obtained by applying the Chvátal-Gomory procedure a finite number of times.

Proof.* We present a proof for the 0-1 case. Thus let $P = \{x \in R^n : Ax \leq b, 0 \leq x \leq 1\} \neq \emptyset, X = P \cap Z^n$, and suppose that $\pi x \leq \pi_0$ with π, π_0 integral is a valid inequality for X . We will show that this inequality is a C-G inequality, or in other words that it can be obtained by applying the Chvátal-Gomory procedure a finite number of times.

Claim 1 The inequality $\pi x \leq \pi_0 + t$ is a valid inequality for P for some $t \in Z_+^1$.

Proof. $z_{LP} = \max\{\pi x : x \in P\}$ is bounded as P is bounded. Take $t = \lceil z_{LP} \rceil - \pi_0$.

Claim 2 There exists a sufficiently large integer M that the inequality

$$\pi x \leq \pi_0 + M \sum_{j \in N^0} x_j + M \sum_{j \in N^1} (1 - x_j) \quad (8.5)$$

is valid for P for every partition (N^0, N^1) of N .

Proof. It suffices to show that the inequality is satisfied at all vertices x^* of P . If $x^* \in Z^n$, then the inequality $\pi x \leq \pi_0$ is satisfied, and so (8.5) is satisfied. Otherwise there exists $\alpha > 0$ such that $\sum_{j \in N^0} x_j^* + \sum_{j \in N^1} (1 - x_j^*) \geq \alpha$ for all partitions (N^0, N^1) of N and all non-integral extreme points x^* of P . Taking $M \geq \frac{t}{\alpha}$, it follows that for all extreme points of P ,

$$\pi x^* \leq \pi_0 + t \leq \pi_0 + M \sum_{j \in N^0} x_j^* + M \sum_{j \in N^1} (1 - x_j^*).$$

Claim 3 If $\pi x \leq \pi_0 + \tau + 1$ is a C-G inequality for X with $\tau \in Z_+^1$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in N^0} x_j + \sum_{j \in N^1} (1 - x_j) \quad (8.6)$$

is a C-G inequality for X .

Proof. Take the inequality $\pi x \leq \pi_0 + \tau + 1$ with weight $(M - 1)/M$ and the inequality (8.5) with weight $1/M$. The resulting C-G inequality is (8.6).

Claim 4 If

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0 \cup \{p\}} x_j + \sum_{j \in T^1} (1 - x_j) \quad (8.7)$$

and

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1 \cup \{p\}} (1 - x_j) \quad (8.8)$$

are C-G inequalities for X where (T^0, T^1) is any partition of $\{1, \dots, p - 1\}$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1} (1 - x_j) \quad (8.9)$$

is a C-G inequality for X .

Proof. Take the inequalities (8.7) and (8.8) with weights $1/2$ and the resulting C-G inequality is (8.9).

Claim 5 If

$$\pi x \leq \pi_0 + \tau + 1$$

is a C-G inequality for X , then

$$\pi x \leq \pi_0 + \tau$$

is a C-G inequality for X .

Proof. Apply Claim 4 successively for $p = n, n - 1, \dots, 1$ with all partitions (T^0, T^1) of $\{1, \dots, p - 1\}$.

Finally starting with $\tau = t - 1$ and using Claim 5 for $\tau = t - 1, \dots, 0$ establishes that $\pi x \leq \pi_0$ is a C-G inequality. ■

For inequalities generated by other arguments, it is sometimes interesting to see how easily they are generated as C-G inequalities, see Exercise 8.15.

Now that we have seen a variety of both ad hoc and general ways to derive valid inequalities, we turn to the important practical question of how to use them.

8.4 A PRIORI ADDITION OF CONSTRAINTS

In discussing branch-and-bound we saw that preprocessing was a first step in tightening a formulation. Here we go a step further. The idea here is to examine the initial formulation $P = \{x : Ax \leq b, x \geq 0\}$ with $X = P \cap Z^n$, find a set of valid inequalities $Qx \leq q$ for X , add these to the formulation immediately giving a new formulation $P' = \{x : Ax \leq b, Qx \leq q, x \geq 0\}$ with $X = P' \cap Z^n$. Then one can apply one's favorite algorithm, Branch-and-Bound or whatever, to formulation P' .

Advantages. One can use standard branch-and-bound software. If the valid inequalities are well chosen so that formulation P' is significantly smaller than P , the bounds should be improved and hence the branch-and-bound algorithm should be more effective. In addition the chances of finding feasible integer solutions in the course of the algorithm should increase.

Disadvantages. Often the family of valid inequalities one would like to add is enormous. In such cases either the linear programs become very big and take a long time to solve, or it becomes impossible to use standard branch-and-bound software because there are too many constraints.

How can one start looking for valid inequalities a priori? In many instances the feasible region X can be written naturally as the intersection of two or more sets with structure, that is, $X = X^1 \cap X^2$. Decomposing or concentrating on one of the sets at a time may be a good idea.

For instance, we may know that the optimization problem over the set $X^2 = P^2 \cap Z^n$ is easy. Then we may be able to find an explicit description of $P^2 = \text{conv}(P^2 \cap Z^n)$. In this case we can replace the initial formulation $P^1 \cap P^2$ by an improved formulation $P' = P^1 \cap P^2$.

Whether the optimization problem over X^2 is easy or not, one may be able to take advantage of the structure to find valid inequalities for X^2 , which allow us to improve its formulation from P^2 to $P'^2 \subset P^2$, and thereby again provide an improved formulation $P' = P^1 \cap P'^2$ for X .

Example 8.8 Uncapacitated Facility Location. Take the "weak" formulation used in the 1950s and 1960s:

$$\begin{aligned} \sum_{j=1}^n x_{ij} &= 1 \text{ for } i = 1, \dots, m \\ \sum_{i=1}^m x_{ij} &\leq my_j \text{ for } j = 1, \dots, n \\ x_{ij} &\geq 0 \text{ for } i = 1, \dots, m, j = 1, \dots, n \\ 0 &\leq y_j \leq 1 \text{ for } j = 1, \dots, n. \end{aligned}$$

Let X_j be the set of points in the polyhedron P_j :

$$\begin{aligned} \sum_{i=1}^m x_{ij} &\leq my_j \\ x_{ij} &\geq 0 \text{ for } i = 1, \dots, m \\ 0 &\leq y_j \leq 1, \end{aligned}$$

with y_j integer. The convex hull P'_j of the set X_j is given by

$$\begin{aligned} x_{ij} &\leq y_j \text{ for } i = 1, \dots, m \\ x_{ij} &\geq 0 \text{ for } i = 1, \dots, m \\ 0 &\leq y_j \leq 1. \end{aligned}$$

Now the reformulation obtained by replacing P_j by P'_j for $j = 1, \dots, n$ is the "strong" formulation P' :

$$\begin{aligned} \sum_{j=1}^n x_{ij} &= 1 \text{ for } i = 1, \dots, m \\ x_{ij} &\leq y_j \text{ for } i = 1, \dots, m, j = 1, \dots, n \\ x_{ij} &\geq 0 \text{ for } i = 1, \dots, m, j = 1, \dots, n \\ 0 \leq y_j &\leq 1 \text{ for } j = 1, \dots, n. \end{aligned}$$

The strong formulation is now commonly used for this problem because the bound it provides is much stronger than that given by the weak formulation, and the linear programming relaxation has solutions that are close to being integral. ■

Example 8.9 Constant Capacity Lot-Sizing. Using the same notation as for ULS introduced in Section 1.5, a basic formulation of the feasible region X is

$$\begin{aligned} s_{t-1} + x_t &= d_t + s_t \text{ for } t = 1, \dots, n \\ x_t &\leq Cy_t \text{ for } t = 1, \dots, n \\ s_0 = s_n = 0, s &\in R_+^{n+1}, x \in R^n, y \in B^n. \end{aligned}$$

We derive two families of inequalities for X . First consider any point $(x, s, y) \in X$. First as $s_{t-1} \geq 0$, the inequality $x_t \leq d_t + s_t$ clearly holds. Along with $x_t \leq Cy_t$ and $y_t \in \{0, 1\}$, it is then not difficult to show that $x_t \leq d_t y_t + s_t$ is valid for X . (Note that without the variable s_t , this is precisely the mixed 0-1 inequality of Example 8.2).

Second, summing the flow conservation constraints, and using $s_t \geq 0$, we get the inequality $\sum_{i=1}^t x_i \geq \sum_{i=1}^t d_i$. Then using $x_i \leq Cy_i$ gives $C \sum_{i=1}^t y_i \geq \sum_{i=1}^t d_i$, or $\sum_{i=1}^t y_i \geq (\sum_{i=1}^t d_i)/C$. Now as $\sum_{i=1}^t y_i$ is integral, we can use Chvátal-Gomory integer rounding to obtain the valid inequality

$$\sum_{i=1}^t y_i \geq \lceil \frac{\sum_{i=1}^t d_i}{C} \rceil.$$

Adding just these $2n$ inequalities significantly strengthens the formulation of this problem. ■

8.5 AUTOMATIC REFORMULATION OR CUTTING PLANE ALGORITHMS

Suppose that $X = P \cap Z^n$ and that we know a family \mathcal{F} of valid inequalities $\pi x \leq \pi_0, (\pi, \pi_0) \in \mathcal{F}$ for X . In many cases \mathcal{F} contains too many inequalities

(2^n or more) for them to be added a priori. Also given a specific objective function, one is not really interested in finding the complete convex hull, but one wants a good approximation to it in the neighborhood of an optimal solution.

We now describe a basic cutting plane algorithm for (IP) , $\max\{cx : x \in X\}$, that generates "useful" inequalities from \mathcal{F} .

Cutting Plane Algorithm

Initialization. Set $t = 0$ and $P^0 = P$.

Iteration t . Solve the linear program:

$$z^t = \max\{cx : x \in P^t\}.$$

Let x^t be an optimal solution.

If $x^t \in Z^n$, stop. x^t is an optimal solution for IP .

If $x^t \notin Z^n$, solve the separation problem for x^t and the family \mathcal{F} .

If an inequality $(\pi^t, \pi_0^t) \in \mathcal{F}$ is found with $\pi^t x^t > \pi_0^t$ so that it cuts off x^t , set $P^{t+1} = P^t \cap \{x : \pi^t x \leq \pi_0^t\}$, and augment t . Otherwise stop.

If the algorithm terminates without finding an integral solution for IP ,

$$P^t = P \cap \{x : \pi^i x \leq \pi_0^i \quad i = 1, \dots, t\}$$

is an improved formulation that can be input to a branch-and-bound algorithm. It should also be noted that in practice it is often better to add several violated cuts at each iteration, and not just one at a time.

In the next section we look at a specific implementation of this algorithm.

8.6 GOMORY'S FRACTIONAL CUTTING PLANE ALGORITHM

Here we consider the integer program:

$$\max\{cx : Ax = b, x \geq 0 \text{ and integer}\}.$$

The idea is to first solve the associated linear programming relaxation and find an optimal basis, choose a basic variable that is not integer, and then generate a Chvátal-Gomory inequality on the constraint associated with this basic variable so as to cut off the linear programming solution. We suppose, given an optimal basis, that the problem is rewritten in the form:

$$\begin{aligned} & \max \bar{a}_{u0} + \sum_{j \in NB} \bar{a}_{uj} x_j \\ & x_{B_u} + \sum_{j \in NB} \bar{a}_{uj} x_j = \bar{a}_{u0} \text{ for } u = 1, \dots, m \\ & x \geq 0 \text{ and integer} \end{aligned}$$

with $\bar{a}_{0j} < 0$ for $j \in NB$ and $\bar{a}_{u0} \geq 0$ for $u = 1, \dots, m$, where NB is the set of nonbasic variables.

If the basic optimal solution x^* is not integer, there exists some row u with $\bar{a}_{u0} \notin Z^1$. Choosing such a row, the Chvátal-Gomory cut for row u is

$$x_{B_u} + \sum_{j \in NB} [\bar{a}_{uj}] x_j \leq [\bar{a}_{u0}]. \tag{8.10}$$

Rewriting this inequality by eliminating x_{B_u} gives

$$\sum_{j \in NB} (\bar{a}_{uj} - [\bar{a}_{uj}]) x_j \geq \bar{a}_{u0} - [\bar{a}_{u0}]$$

or

$$\sum_{j \in NB} f_{uj} x_j \geq f_{u0} \tag{8.11}$$

where $f_{uj} = \bar{a}_{uj} - [\bar{a}_{uj}]$ for $j \in NB$, and $f_{u0} = \bar{a}_{u0} - [\bar{a}_{u0}]$.

By the definitions and the choice of row u , $0 \leq f_{uj} < 1$ and $0 < f_{u0} < 1$. As $x_j^* = 0$ for all nonbasic variables $j \in NB$ in the optimal LP solution, this inequality cuts off x^* . It is also important to observe that the difference between the left- and right-hand sides of the Chvátal-Gomory inequality (8.10), and hence also of (8.11), is integral when x is integral, so that when (8.11) is rewritten as an equation:

$$s = -f_{u0} + \sum_{j \in NB} f_{uj} x_j,$$

the slack variable s is a nonnegative integer variable.

Example 8.10 Consider the integer program

$$\begin{aligned} z = \max \quad & 4x_1 - x_2 \\ & 7x_1 - 2x_2 \leq 14 \\ & \quad \quad x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x_1, \quad x_2 \geq 0 \text{ and integer.} \end{aligned}$$

Adding slack variables x_3, x_4, x_5 , observe that as the constraint data is integer, the slack variables must also take integer values. Now solving as a linear program gives:

$$\begin{aligned} z = \max \frac{59}{7} \quad & -\frac{4}{7}x_3 - \frac{1}{7}x_4 \\ x_1 \quad & +\frac{1}{7}x_3 + \frac{2}{7}x_4 = \frac{20}{7} \\ x_2 \quad & \quad \quad +x_4 = 3 \\ & -\frac{2}{7}x_3 + \frac{10}{7}x_4 + x_5 = \frac{23}{7} \\ x_1, x_2, x_3, x_4, x_5 \quad & \geq 0 \text{ and integer.} \end{aligned}$$

The optimal linear programming solution is $x = (\frac{20}{7}, 3, 0, 0, \frac{23}{7}) \notin Z_+^5$, so we use the first row, in which the basic variable x_1 is fractional, to generate the cut:

$$\frac{1}{7}x_3 + \frac{2}{7}x_4 \geq \frac{6}{7}$$

or

$$s = -\frac{6}{7} + \frac{1}{7}x_3 + \frac{2}{7}x_4$$

with $s, x_3, x_4 \geq 0$ and integer.

Adding this cut, and reoptimizing leads to the new optimal tableau

$$z = \max \frac{15}{2}$$

x_1	$-\frac{1}{2}x_5$	$-3s$		$= 2$
x_2	$-\frac{1}{2}x_5$	$+s$		$= \frac{1}{2}$
x_3	$-x_5$	$-5s$		$= 1$
x_4	$+\frac{1}{2}x_5$	$+6s$		$= \frac{5}{2}$
$x_1, x_2, x_3, x_4, x_5, s$			≥ 0 and integer.	

Now the new optimal linear programming solution $x = (2, \frac{1}{2}, 1, \frac{5}{2}, 0)$ is still not integer, as the original variable x_2 , and the slack variable x_4 are fractional. The Gomory fractional cut on row 2, in which x_2 is basic, is $\frac{1}{2}x_5 \geq \frac{1}{2}$ or $-\frac{1}{2}x_5 + t = -\frac{1}{2}$ with $t \geq 0$ and integer. Adding this constraint and reoptimizing, we obtain

$$z = \max 7$$

x_1		$-3s$	$-t$		$= 2$
x_2		$+s$			$= 1$
x_3		$+s$	$-t$		$= 1$
x_4		$-5s$	$-2t$		$= 2$
x_5		$+6s$	$+t$		$= 2$
$x_1, x_2, x_3, x_4, x_5, s, t$			≥ 0 and integer.		$= 1$

Now the linear programming solution is integral, and optimal, and thus $(x_1, x_2) = (2, 1)$ solves the original integer program. ■

It is natural to also look at the cuts in the space of the original variables.

Example 8.10 (cont) Considering the first cut, and substituting for x_3 and x_4 gives:

$$\frac{1}{7}(14 - 7x_1 + 2x_2) + \frac{2}{7}(3 - x_2) \geq \frac{6}{7}$$

or $x_1 \leq 2$.

In Figure 8.2 we can verify that this inequality is valid and cuts off the fractional solution $(\frac{20}{7}, 3)$. Similarly, substituting for x_5 in the second cut $\frac{1}{2}x_5 \geq \frac{1}{2}$ gives the valid inequality $x_1 - x_2 \leq 1$ in the original variables. ■

To find a general formula that gives us the cut in terms of the original variables, one can show:

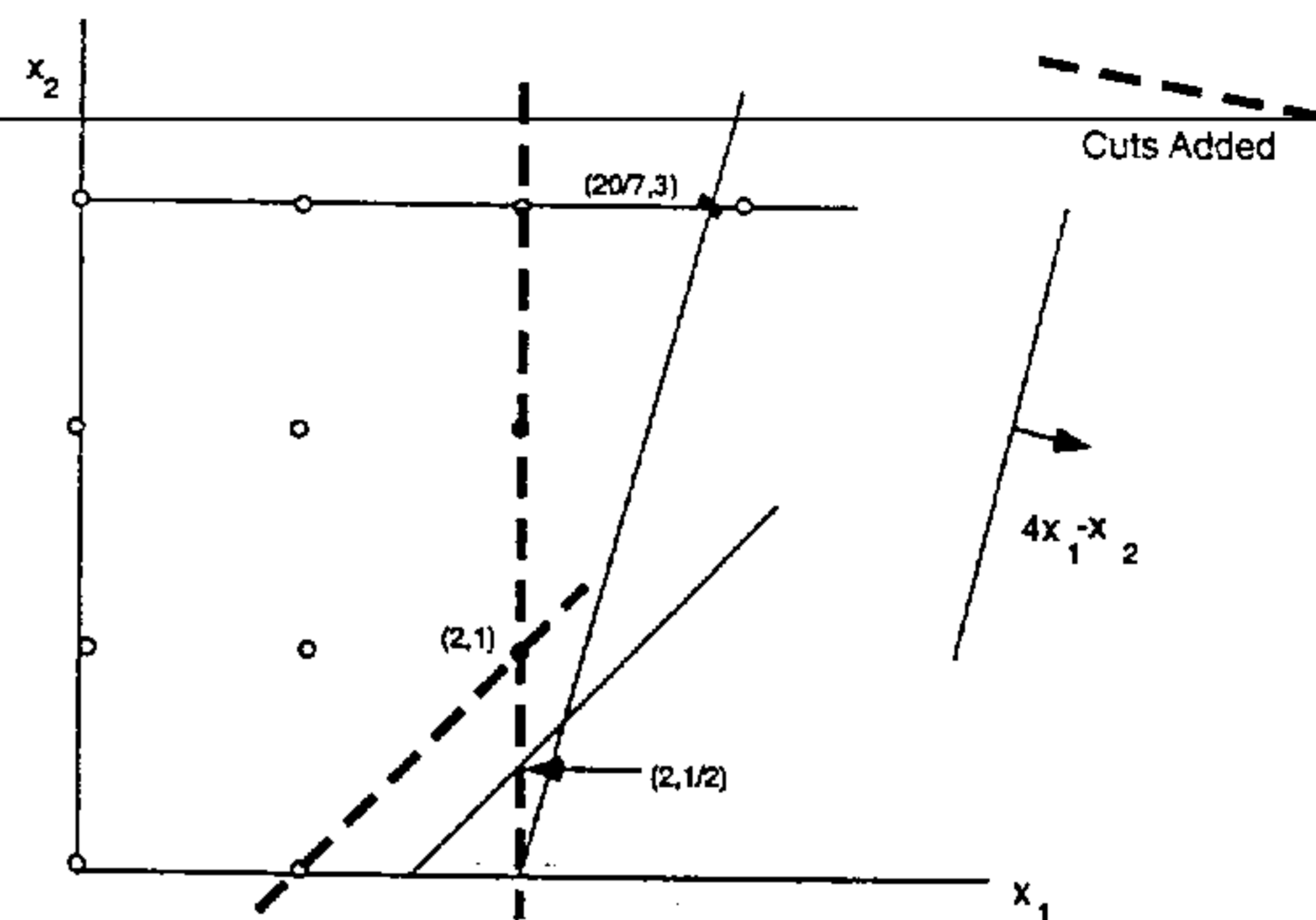


Fig. 8.2 Gomory cutting planes

Proposition 8.5 Let β be row u of B^{-1} , and $q_i = \beta_i - [\beta_i]$ for $i = 1, \dots, m$. The Gomory cut $\sum_{j \in NB} f_{uj}x_j \geq f_{u0}$, when written in terms of the original variables, is the Chvátal-Gomory inequality

$$\sum_{j=1}^n [qa_j]x_j \leq [qb].$$

Looking at the first Gomory cut generated in Example 8.10, β is given by the coefficients of the slack variables in row $u = 1$, so $\beta = (\frac{1}{7}, \frac{2}{7}, 0)$. Thus $q = (\frac{1}{7}, \frac{2}{7}, 0)$ and we obtain $1x_1 + 0x_2 \leq [\frac{20}{7}] = 2$.

8.7 MIXED INTEGER CUTS

8.7.1 The Basic Mixed Integer Inequality

We saw above that when $y \leq b, y \in Z^1$, the rounding inequality $y \leq [b]$ suffices to generate all the inequalities for a pure integer program. Here we examine if there is a similar basic inequality for mixed integer programs.

Proposition 8.6 Let $X \geq = \{(x, y) \in R_+^1 \times Z^1 : x + y \geq b\}$, and $f = b - [b] > 0$. The inequality

$$x \geq f([b] - y) \text{ or } \frac{x}{f} + y \geq [b]$$

is valid for $X \geq$.

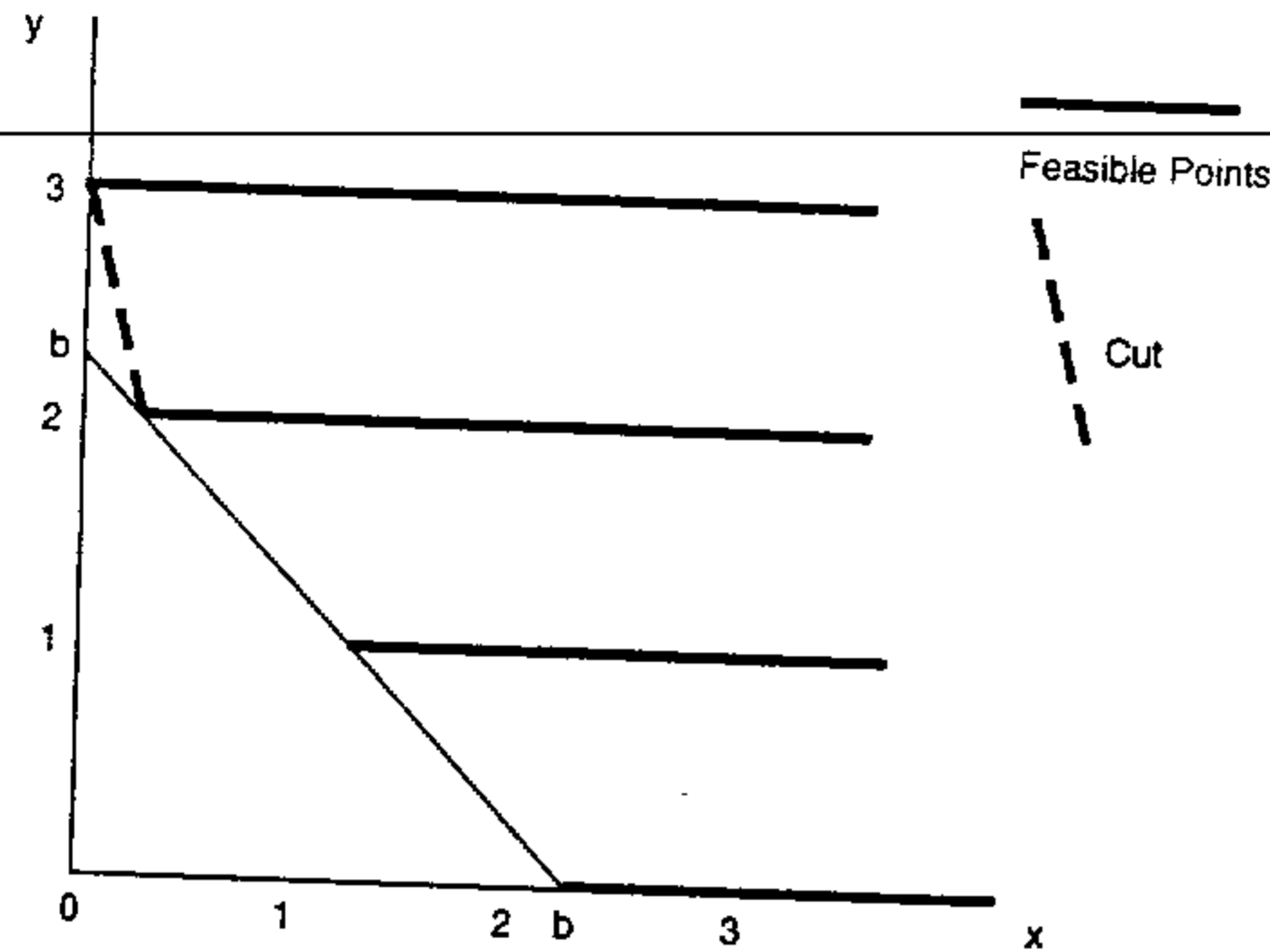


Fig. 8.3 Basic mixed inequality

Proof. If $y \geq [b]$, then $x \geq 0 \geq f([b] - y)$. If $y < [b]$, then

$$\begin{aligned} x &\geq b - y = f + ([b] - y) \\ &\geq f + f([b] - y), \text{ as } [b] - y \geq 0 \text{ and } f < 1, \\ &= f([b] - y). \end{aligned}$$

The situation is shown in Figure 8.3. The following corollary allows us to compare more directly with the all-integer case.

Corollary If $X \leq = \{(x, y) \in R_+^1 \times Z^1 : y \leq b + x\}$, and $f = b - [b] > 0$, the inequality

$$y \leq [b] + \frac{x}{1-f}$$

is valid for $X \leq$.

Proof. Rewriting $y \leq b + x$ as $x - y \geq -b$ and observing that $-b - [-b] = 1 - f$, we obtain from Proposition 8.6 that $\frac{x}{1-f} - y \geq [-b] = -[b]$.

Thus we see that when the continuous variable $x = 0$, we obtain the basic integer rounding inequality.

Example 8.6 (cont) The trucking example discussed earlier led to the set $2y_1 + 2y_2 + y_3 + y_4 + \frac{s}{11} \geq \frac{72}{11}$ with $y \in Z_+^4$ and $s \geq 0$. Using Proposition 8.6

with $[b] = 7$ and $f = \frac{6}{11}$, we obtain immediately that

$$\frac{s}{11} \geq \frac{6}{11}(7 - 2y_1 - 2y_2 - y_3 - y_4)$$

is a valid inequality. ■

8.7.2 The Mixed Integer Rounding (MIR) Inequality

To obtain a slight variant of the basic inequality, we consider a set

$$X^{MIR} = \{(x, y) \in R_+^1 \times Z_+^2 : a_1 y_1 + a_2 y_2 \leq b + x\},$$

where a_1, a_2 and b are scalars with $b \notin Z^1$.

Proposition 8.7 Let $f = b - [b]$ and $f_i = a_i - [a_i]$ for $i = 1, 2$. Suppose $f_1 \leq f \leq f_2$, then

$$[a_1]y_1 + ([a_2] + \frac{f_2 - f}{1 - f})y_2 \leq [b] + \frac{x}{1 - f} \quad (8.12)$$

is valid for X^{MIR} .

Proof. $(x, y) \in X^{MIR}$ satisfies $[a_1]y_1 + [a_2]y_2 \leq b + x + (1 - f_2)y_2$ as $y_1 \geq 0$, and $a_2 = [a_2] - (1 - f_2)$. Now the Corollary to Proposition 8.6 gives

$$[a_1]y_1 + [a_2]y_2 \leq [b] + [x + (1 - f_2)y_2] / (1 - f),$$

which is the required inequality. ■

Example 8.11 Consider the set $X = \{(y, x) \in Z_+^3 \times R_+^1 : \frac{10}{3}y_1 + 1y_2 + \frac{11}{4}y_3 \leq \frac{21}{2} + x\}$. Using Proposition 8.7, we have $f = 1/2, f_1 = 1/3, f_2 = 0, f_3 = 3/4$, and thus

$$3y_1 + y_2 + \frac{5}{2}y_3 \leq 10 + 2x$$

is valid for X . ■

8.7.3 The Gomory Mixed Integer Cut*

Here we continue to consider mixed integer programs. As for all integer programs in Section 8.6, any row of an optimal linear programming tableau, in which an integer variable is basic but fractional, can be used to generate a cut removing the optimal linear programming solution. Specifically, such a row leads to a set of the form:

$$X^G = \{(y_B, y, x) \in Z^1 \times Z_+^{n_1} \times R_+^{n_2} : y_B, u + \sum_{j \in N_1} \bar{a}_{uj} y_j + \sum_{j \in N_2} \bar{a}_{uj} x_j = \bar{a}_{u0}\},$$

where $n_i = |N_i|$ for $i = 1, 2$.

Proposition 8.8 If $\bar{a}_{u0} \notin Z^1$, $f_j = \bar{a}_{uj} - [\bar{a}_{uj}]$ for $j \in N_1 \cup N_2$, and $f_0 = \bar{a}_{u0} - [\bar{a}_{u0}]$, the Gomory mixed integer cut

$$\sum_{f_j \leq f_0} f_j y_j + \sum_{f_j > f_0} \frac{f_0(1-f_j)}{1-f_0} y_j + \sum_{\bar{a}_{uj} > 0} \bar{a}_{uj} x_j + \sum_{\bar{a}_{uj} < 0} \frac{f_0}{1-f_0} \bar{a}_{uj} x_j \geq f_0$$

is valid for X^G .

Proof. The mixed integer rounding inequality (8.12) for X^G is

$$y_{B_u} + \sum_{f_j \leq f_0} [\bar{a}_{uj}] y_j + \sum_{f_j > f_0} ([\bar{a}_{uj}] + \frac{f_j - f_0}{1-f_0}) y_j + \sum_{\bar{a}_{uj} < 0} \frac{\bar{a}_{uj}}{1-f_0} x_j \leq [\bar{a}_{u0}].$$

Substituting for y_{B_u} proves the claim. ■

Example 8.12 Consider the mixed integer program:

$$\begin{aligned} z = \max \quad & 4x_1 - x_2 \\ & 7x_1 - 2x_2 \leq 14 \\ & \quad \quad x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x_1 \in Z_+^1, \quad x_2 \geq 0. \end{aligned}$$

Note that this is the same as Example 8.10 except that variable $x_2 \in R_+^1$ is a real variable. Solving as a linear program gives:

$$\begin{aligned} z = \max \frac{59}{7} \quad & -\frac{4}{7}x_3 - \frac{1}{7}x_4 \\ x_1 \quad & +\frac{1}{7}x_3 + \frac{2}{7}x_4 = \frac{20}{7} \\ x_2 \quad & \quad \quad +x_4 = 3 \\ & -\frac{2}{7}x_3 + \frac{10}{7}x_4 + x_5 = \frac{23}{7} \\ x_1 \in Z_+^1, \quad & x_2, x_3, x_4, x_5 \geq 0. \end{aligned}$$

The basic variable x_1 is fractional and the first row gives the MIR cut $x_1 \leq 2$, which after elimination of x_1 becomes the Gomory mixed integer cut:

$$\frac{1}{7}x_3 + \frac{2}{7}x_4 \geq \frac{6}{7}.$$

Adding this cut and reoptimizing leads to the solution $x = (2, \frac{1}{2})$, which is feasible and hence optimal for the mixed integer program. This can also be seen graphically in Figure 8.2 with just the addition of the cut $x_1 \leq 2$. ■

8.8 DISJUNCTIVE INEQUALITIES*

The set $X = X^1 \cup X^2$ with $X^i \subseteq R_+^n$ for $i = 1, 2$ is a *disjunction* (union) of the two sets X^1 and X^2 . The following simple result has already been used implicitly in Proposition 8.6 in deriving the basic mixed integer inequality.

Proposition 8.9 If $\sum_{j=1}^n \pi_j^i x_j \leq \pi_0^i$ is valid for X^i for $i = 1, 2$, then the inequality

$$\sum_{j=1}^n \pi_j x_j \leq \pi_0$$

is valid for X if $\pi_j \leq \min[\pi_j^1, \pi_j^2]$ for $j = 1, \dots, n$ and $\pi_0 \geq \max[\pi_0^1, \pi_0^2]$.

Proof. If $x \in X$, then $x \in X^1$ or $x \in X^2$. If $x \in X^i$, then as $x \geq 0$, $\sum_{j=1}^n \pi_j x_j \leq \sum_{j=1}^n \pi_j^i x_j \leq \pi_0^i \leq \pi_0$ for $i = 1, 2$. Thus the inequality is valid for all $x \in X$. ■

Disjunctions of polyhedra are particularly interesting. Modeling such sets is easy; see Exercises 1.3 and 8.10. Using Proposition 8.2, it is also easy to characterize valid inequalities for such disjunctions.

Proposition 8.10 If $P^i = \{x \in R_+^n : A^i x \leq b^i\}$ for $i = 1, 2$ are nonempty polyhedra, then (π, π_0) is a valid inequality for $\text{conv}(P^1 \cup P^2)$ if and only if there exist $u^1, u^2 \geq 0$ such that $\pi \leq u^i A^i$ and $\pi_0 \geq u^i b^i$ for $i = 1, 2$.

Example 8.13 Let $P^1 = \{x \in R_+^2 : -x_1 + x_2 \leq 1, x_1 + x_2 \leq 5\}$ and $P^2 = \{x \in R_+^2 : x_2 \leq 4, -2x_1 + x_2 \leq -6, x_1 - 3x_2 \leq -2\}$. Taking $u^1 = (2, 1)$ and $u^2 = (\frac{5}{2}, \frac{1}{2}, 0)$ and applying Proposition 8.10 gives that $-x_1 + 3x_2 \leq 7$ is valid for $P^1 \cup P^2$. See Figure 8.4. ■

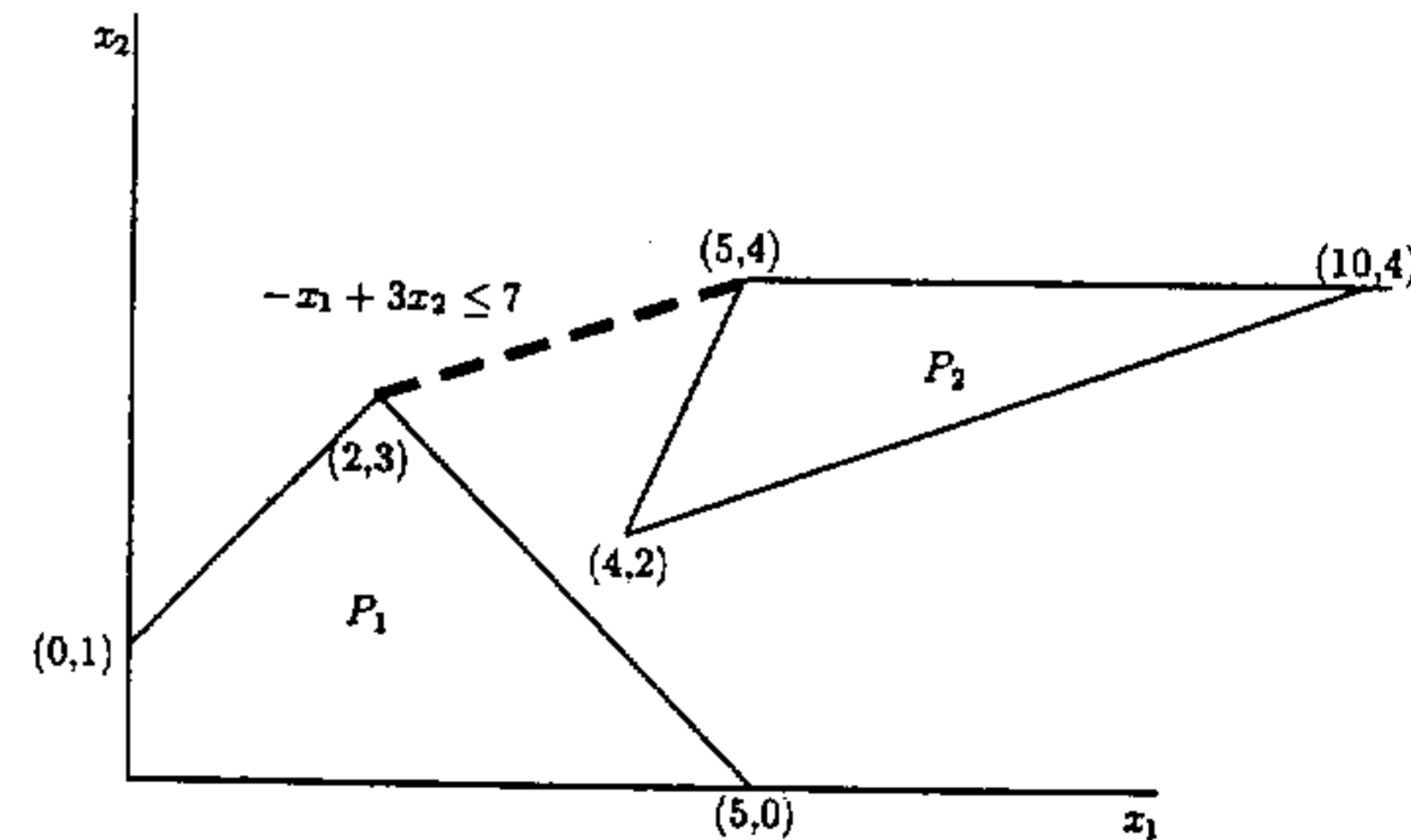


Fig. 8.4 Disjunctive inequality

Specializing further, we consider 0-1 problems, where $X = P \cap Z^n \subseteq \{0, 1\}^n$ with $P = \{x \in R^n : Ax \leq b, 0 \leq x \leq 1\}$. Let $P^0 = P \cap \{x \in R^n : x_j = 0\}$, and $P^1 = P \cap \{x \in R^n : x_j = 1\}$ for some $j \in \{1, \dots, n\}$.

Proposition 8.11 The inequality (π, π_0) is a valid for $\text{conv}(P^0 \cup P^1)$ if there exists $u^i \in R_+^m, v^i \in R_+^n, w^i \in R_+^1$ for $i = 0, 1$ such that

$$\begin{aligned} \pi &\leq u^0 A + v^0 + w^0 e_j, & \pi &\leq u^1 A + v^1 - w^1 e_j, \\ \pi_0 &\geq u^0 b + 1 \cdot v^0, & \pi_0 &\geq u^1 b + 1 \cdot v^1 - w^1. \end{aligned}$$

Proof. Apply Proposition 8.10 with $P^0 = \{x \in R_+^n : Ax \leq b, x \leq 1, x_j \leq 0\}$ and $P^1 = \{x \in R_+^n : Ax \leq b, x \leq 1, -x_j \leq -1\}$. ■

Example 8.14 Consider the 0-1 knapsack problem

$$\begin{aligned} \max & 12x_1 + 14x_2 + 7x_3 + 12x_4 \\ & 4x_1 + 5x_2 + 3x_3 + 6x_4 \leq 8 \\ & x \in B^4 \end{aligned}$$

with linear programming solution $x^* = (1, 0.8, 0, 0)$.

As $x_2^* = 0.8$ is fractional, we choose $j = 2$ in defining P^0 and P^1 , and then look for the most violated valid inequality (π, π_0) given by Proposition 8.11. To do this, we solve a linear program consisting of maximizing $\pi x^* - \pi_0$ over the polyhedron describing the coefficients of the valid inequalities given in the proposition, namely

$$\begin{aligned} \max & 1.0\pi_1 + 0.8\pi_2 - \pi_0 \\ & \pi_1 \leq 4u^0 + v_1^0, \pi_1 \leq 4u^1 + v_1^1 \\ & \pi_2 \leq 5u^0 + v_2^0 + w^0, \pi_2 \leq 5u^1 + v_2^1 - w^1 \\ & \pi_3 \leq 3u^0 + v_3^0, \pi_3 \leq 3u^1 + v_3^1 \\ & \pi_4 \leq 6u^0 + v_4^0, \pi_4 \leq 6u^1 + v_4^1 \\ & \pi_0 \geq 8u^0 + v_1^0 + v_2^0 + v_3^0 + v_4^0 \\ & \pi_0 \geq 8u^1 + v_1^1 + v_2^1 + v_3^1 + v_4^1 - w^1 \\ & u^0, u^1, v^0, v^1, w^0, w^1 \geq 0. \end{aligned}$$

Note that for the linear program to have a bounded optimal value, it is necessary to normalize the inequality. Two possibilities are $\sum_{j=1}^n \pi_j \leq 1$ or $\pi_0 = 1$. The resulting inequality is

$$1x_1 + \frac{1}{4}x_2 \leq 1,$$

with violation of $\frac{1}{5}$. For P^0 , it is a combination of constraints $x_1 \leq 1$ and $x_2 \leq 0$ with $v_1^0 = 1$ and $w^0 = \frac{1}{4}$ respectively. For P^1 , it is a combination of the knapsack inequality $4x_1 + 5x_2 + 3x_3 + 6x_4 \leq 8$ and $-x_2 \leq -1$ with $u^1 = \frac{1}{4}$ and $w^1 = 1$ respectively. Both normalizations lead to the same inequality. ■

The idea of looking for the most violated inequality will be pursued in the next chapter, when we try to obtain "strong" inequalities.

8.9 NOTES

8.2 The inequality (8.3), called a *blossom inequality*, is from [Edm65a].

8.3 The rounding procedure to generate cuts is from [Gom58]. The general procedure described here and the proof of Theorem 8.4 for bounded integer programs is from [Chv73]. In [Sch80], the result is extended to unbounded polyhedra.

8.4 The strong formulation for *UFL* is used computationally in [Spi69].

8.5 The first cutting plane algorithm reported is the procedure used to solve a 54-city *TSP* in [DanFulJoh54].

8.6 The fractional cutting plane algorithm is presented in [Gom58], [Gom63]. The latter paper also contains a beautiful theoretical result, namely that the algorithm converges finitely if the rows off which the cuts are generated are properly chosen.

8.7 Gomory mixed integer cuts are proposed in [Gom60]. The presentation of mixed integer rounding inequalities is from [NemWol90]. The theory of superadditive valid inequalities and superadditive duality [Joh80] and Chapter II.1 in [NemWol88] provides a complete explanation of cuts for integer and mixed integer programs.

Gomory has shown that finite convergence can be attained with his mixed integer cuts if the objective function is integer valued. It is an open question whether this is true for 0-1 mixed integer programs with an arbitrary objective function. In [CooKanSch90], the question of how finite convergence might be obtained is reexamined.

Recently [GunPoc98] present a new way to combine basic mixed integer inequalities. Gomory mixed integer cuts have also been recently revived as a computational tool; see [Balasetal96].

8.8 Disjunctive and Gomory mixed integer cuts are closely related. Proposition 8.9 was already used implicitly by Gomory in developing the mixed integer cut. In the same way that the Chvátal-Gomory procedure can be used to generate all valid inequalities for an integer program, it can be shown that a simple disjunctive procedure repeated finitely (see Exercise 8.11) can be used to generate all valid inequalities for a 0-1 mixed integer program.

The approach here is based on the disjunction of polyhedra developed by Balas [Balas75a] in the 1970s; see also [Jer72]. In particular, Balas shows the beautiful result that to obtain the convex hull of a 0-1 *MIP*, it suffices to take the convex hulls of each 0-1 variable one at a time. Related to this result, a variety of extended formulations have recently been proposed to obtain tighter formulations for 0-1 *MIPs* [LovSch91], [BalasCerCor93], [SheAda90].

In practice, under the name *lift and project*, Proposition 8.11 is used as in Example 8.14 to develop a disjunctive cutting plane algorithm. Computational results are given in [BalasCerCor96].

8.10 EXERCISES

1. For each of the three sets below, find a missing valid inequality and verify graphically that its addition to the formulation gives $\text{conv}(X)$.

(i) $X = \{x \in B^2 : 3x_1 - 4x_2 \leq 1\}$

(ii) $X = \{(x, y) \in R_+^1 \times B^1 : x \leq 20y, x \leq 7\}$

(iii) $X = \{(x, y) \in R_+^1 \times Z_+^1 : x \leq 6y, x \leq 16\}$

2. In each of the examples below a set X and a point x or (x, y) are given. Find a valid inequality for X cutting off the point.

(i) $X = \{(x, y) \in R_+^2 \times B^1 : x_1 + x_2 \leq 2y, x_j \leq 1 \text{ for } j = 1, 2\}$

$(x_1, x_2, y) = (1, 0, 0.5)$

(ii) $X = \{(x, y) \in R_+^1 \times Z_+^1 : x \leq 9, x \leq 4y\}$

$(x, y) = (9, \frac{9}{4})$

(iii) $X = \{(x_1, x_2, y) \in R_+^2 \times Z_+^1 : x_1 + x_2 \leq 25, x_1 + x_2 \leq 8y\}$

$(x_1, x_2, y) = (20, 5, \frac{25}{8})$

(iv) $X = \{x \in Z_+^5 : 9x_1 + 12x_2 + 8x_3 + 17x_4 + 13x_5 \geq 50\}$

$x = (0, \frac{25}{6}, 0, 0, 0)$

(v) $X = \{x \in Z_+^4 : 4x_1 + 8x_2 + 7x_3 + 5x_4 \leq 33\}$

$x = (0, 0, \frac{33}{7}, 0)$

3. Prove that $y_2 + y_3 + 2y_4 \leq 6$ is valid for

$X = \{y \in Z_+^4 : 4y_1 + 5y_2 + 9y_3 + 12y_4 \leq 34\}$.

4. Consider the problem

$$\begin{aligned} \min & x_1 + 2x_2 \\ & x_1 + x_2 \geq 4 \\ & \frac{1}{2}x_1 + \frac{5}{2}x_2 \geq \frac{15}{2} \\ & x \in Z_+^2. \end{aligned}$$

Show that $x^* = (\frac{15}{4}, \frac{1}{4})$ is the optimal linear programming solution and find an inequality cutting off x^* .

5. Solve $\min\{5x_1 + 9x_2 + 23x_3 : 20x_1 + 35x_2 + 95x_3 \geq 319, x \in Z_+^3\}$ using Chvátal-Gomory inequalities or Gomory's cutting plane algorithm.

6. Solve $\max\{5x_1 + 9x_2 + 23x_3 - 4s : 2x_1 + 3x_2 + 9x_3 \leq 32 + s, x \in Z_+^3, s \in R_+^1\}$ using MIR inequalities.

7. (i) Show that the inequality $x_t \leq d_t y_t + s_t$ is valid for ULS .

(ii) Show that $x_t + x_{t+1} \leq (d_t + d_{t+1})y_t + d_{t+1}y_{t+1} + s_{t+1}$ is valid.

(iii) For $l \leq n$, $L = \{1, \dots, l\}$ and $S \subseteq L$, show that the inequality

$$\sum_{j \in S} x_j \leq \sum_{j \in S} \left(\sum_{t=j}^l d_t \right) y_j + s_l$$

is valid for ULS .

8. Consider the stable set problem. An *odd hole* is a cycle with an odd number of nodes and no edges between nonadjacent nodes of the cycle. Show that if H is the node set of an odd hole,

$$\sum_{j \in H} x_j \leq (|H| - 1)/2$$

is a valid inequality.

9. Use the mixed integer rounding procedure to show that

$$(y_1 + 6y_2)/4 + y_3 + 4y_4 \geq 16$$

is a valid inequality for

$$X = \{y \in Z_+^4 : y_1 + 6y_2 + 12y_3 + 48y_4 \geq 184\}.$$

10. Use the mixed integer rounding procedure to show that

$$x_2 + x_4 \leq 20 + 4(y - 2)$$

is a valid inequality for $X =$

$$\{(x, y) \in R_+^4 \times Z_+^1 : x_1 + x_2 + x_3 + x_4 \leq 10y, x_1 \leq 13, x_2 \leq 15, x_3 \leq 6, x_4 \leq 9\}.$$

11.(i) Show that if $\pi x \leq \pi_0 + \alpha(x_j - k)$ and $\pi x \leq \pi_0 + \beta(k + 1 - x_j)$ with $\alpha, \beta > 0$ and $k \in Z^1$ are both valid for a polyhedron P , then $\pi x \leq \pi_0$ is valid for $P \cap \{x : x_j \in Z^1\}$. An inequality generated in this way is called a *D-inequality*.

(ii)* Show that if $P \subseteq R^n$ is a polyhedron and $j \in \{1, \dots, n\}$, every valid inequality for $\text{conv}(P \cap \{x : x_j \in B^1\})$ is or is dominated by a *D-inequality*.

12.* Prove that if $P^k = \{x \in R^n : A^k x \leq b^k\}$ are bounded polyhedra for $k = 1, 2$, then $\text{conv}(P^1 \cup P^2) = \{x : \text{there exists } (x, z^1, z^2, y^1, y^2) \in R^n \times R^n \times R^n \times R_+^1 \times R_+^1 \text{ satisfying } A^k z^k \leq b^k, y^k \leq x - z^k, y^1 + y^2 = 1\}$.

13. Consider an instance of the generalized transportation problem

$$\begin{aligned} \min & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} \leq b_i \text{ for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \geq d_j \text{ for } j = 1, \dots, n \\ & x \in Z_+^{mn} \end{aligned}$$

with $m = 4, n = 6, a = (15, 25, 40, 70), b = (10, 5, 7, 4), d = (45, 120, 165, 214, 64, 93)$ and

$$(c_{ij}) = \begin{pmatrix} 23 & 12 & 34 & 25 & 27 & 16 \\ 29 & 24 & 43 & 35 & 28 & 19 \\ 43 & 31 & 52 & 36 & 30 & 21 \\ 54 & 36 & 54 & 46 & 34 & 27 \end{pmatrix}$$

Solve with a mixed integer programming system. Now by inspecting the linear programming solution, or otherwise, add one or more valid inequalities to the formulation and resolve. Compare the number of nodes in the branch-and-bound tree before and after. Try to minimize the number of nodes.

14. Consider a telecommunications problem where the demands between pairs of nodes are given. The problem is to install sufficient capacity on the edges of the graph so that all the demands can be satisfied simultaneously. If there is flow of one demand type from i to j , and simultaneously others from j to i , the capacity available must be the sum of the two opposite flows. Capacity can be installed in units of 1 and/or 24, costing 1 and 10 respectively. For a graph on 6 nodes, the following demand matrix must be satisfied

$$(d_{ij}) = \begin{pmatrix} . & 12 & 51 & - & - & . \\ . & . & 53 & 51 & - & . \\ . & . & . & - & 32 & . \\ . & . & . & . & 91 & . \\ . & . & . & . & . & . \end{pmatrix}$$

Formulate and solve with a mixed integer programming system. Try to tighten the formulation.

15. (i) Derive the inequalities of Example 8.1 as C-G inequalities.

(ii) Consider the set $X = \{x \in B^4 : x_i + x_j \leq 1 \text{ for all } 1 \leq i < j \leq 4\}$. Derive the clique inequalities $x_1 + x_2 + x_3 \leq 1$ and $x_1 + x_2 + x_3 + x_4 \leq 1$ as C-G inequalities.

9

Strong Valid Inequalities

INTRODUCTION

In the last chapter we have seen a variety of valid inequalities, and presented a generic cutting plane algorithm. The Gomory fractional cutting plane algorithm is a special case of this algorithm with the particularity that finding cuts is very easy at each iteration. Theoretically it is of interest because it has been shown to terminate after a finite number of iterations, but in practice it has not been successful. However, Gomory mixed integer cuts, as well as the Gomory and Johnson's conjunctive cuts, have been recently successfully used in practice.

Here we address the question of finding *strong* valid inequalities that are hopefully even more effective. The basic cutting plane algorithm is the same as in Section 8.5, however:

We need to say what "strong" means — for our purposes it is any inequality that leads to a stronger formulation. However, in Section 9.2 (optional!) we formalize what is meant by the strength of an inequality.

I) Describing interesting families \mathcal{F} of strong inequalities may be far from easy.

II) Given a family \mathcal{F} of strong valid inequalities, the separation problem for \mathcal{F} may require a lot of work. It may be polynomially solvable, or it may be NP-hard, in which case a heuristic algorithm has to be developed. In Sections 9.3–9.5, we examine three sets: 0-1 knapsack sets, mixed 0-1 sets, and the set of incidence vectors of subtours that arises in a generalization of the

traveling salesman problem. We develop a family of strong valid inequalities, and discuss the resulting separation problem for each of the families.

(iv) To solve difficult or large problems to optimality, strong cutting planes need to be embedded into a branch-and-bound framework. The resulting algorithms, called branch-and-cut algorithms, are discussed in Section 9.6.

In discussing separation algorithms, it is important to remember certain ideas encountered earlier. First because Efficient (Polynomial) Optimization and Efficient Separation are equivalent,

(i) If Efficient Optimization holds for a class of problems of the form $\max\{cx \mid x \in X\}$, it may be possible to obtain an "explicit" description of the convex hull of X , and perhaps also a combinatorial separation algorithm for $\text{conv}(X)$, and

(ii) If the Optimization Problem is \mathcal{NP} -hard, there is no hope (unless $\mathcal{P} = \mathcal{NP}$) of obtaining an explicit description of $\text{conv}(X)$, but this should not deter us from looking for families of strong valid inequalities.

As before, the idea of *decomposition* may be useful. In particular,

(iii) If we can break up the feasible set so that $X = X^1 \cap X^2$ where the optimization problem over X^2 is polynomially solvable, then we can attempt to find $\text{conv}(X^2)$, and

(iv) If $X = X^1 \cap X^2$, but the optimization problems over X^1 and X^2 are both \mathcal{NP} -hard, it may be still be worthwhile (and easier) to attempt to find valid inequalities for X^1 and X^2 separately in the hope that the resulting inequalities will also be strong for the intersection $X = X^1 \cap X^2$.

9.2 STRONG INEQUALITIES

Here we address briefly the question of what it means for an inequality to be strong for a set $P = \{x \in R_+^n : Ax \leq b\}$. This leads us to introduce certain concepts important for the description of polyhedra. We also present different arguments that can be used to prove that an inequality is strong, or to show that a set of inequalities describes the convex hull of a discrete set.

9.2.1 Dominance

We note first that the inequalities $\pi x \leq \pi_0$ and $\lambda \pi x \leq \lambda \pi_0$ are identical for any $\lambda > 0$.

Definition 9.1 If $\pi x \leq \pi_0$ and $\mu x \leq \mu_0$ are two valid inequalities for $P \subseteq R_+^n$, $\pi x \leq \pi_0$ *dominates* $\mu x \leq \mu_0$ if there exists $u > 0$ such that $\pi \geq u\mu$ and $\pi_0 \leq u\mu_0$, and $(\pi, \pi_0) \neq (u\mu, u\mu_0)$.

Observe that if $\pi x \leq \pi_0$ dominates $\mu x \leq \mu_0$, then $\{x \in R_+^n : \pi x \leq \pi_0\} \subseteq \{x \in R_+^n : \mu x \leq \mu_0\}$.

Definition 9.2 A valid inequality $\pi x \leq \pi_0$ is *redundant* in the description of P , if there exist $k \geq 1$ valid inequalities $\pi^i x \leq \pi_0^i$ for $i = 1, \dots, k$ for and weights $u_i > 0$ for $i = 1, \dots, k$ such that $(\sum_{i=1}^k u_i \pi^i) x \leq \sum_{i=1}^k u_i \pi_0^i$ dominates $\pi x \leq \pi_0$.

For example, we observe that $\{x \in R_+^n : \pi^i x \leq \pi_0^i \text{ for } i = 1, \dots, k\} \subseteq \{x \in R_+^n : (\sum_{i=1}^k u_i \pi^i) x \leq \sum_{i=1}^k u_i \pi_0^i\} \subseteq \{x \in R_+^n : \pi x \leq \pi_0\}$.

Example 9.1 Taking $n = 2$, $(\pi, \pi_0) = (1, 3, 4)$ and $(\mu, \mu_0) = (2, 4, 9)$, we see that with $u = \frac{1}{2}$, $\pi \geq \frac{1}{2}\mu$ and $\pi_0 \leq \frac{1}{2}\mu_0$, and so $x_1 + 3x_2 \leq 4$ dominates $2x_1 + 4x_2 \leq 9$. See Figure 9.1a.

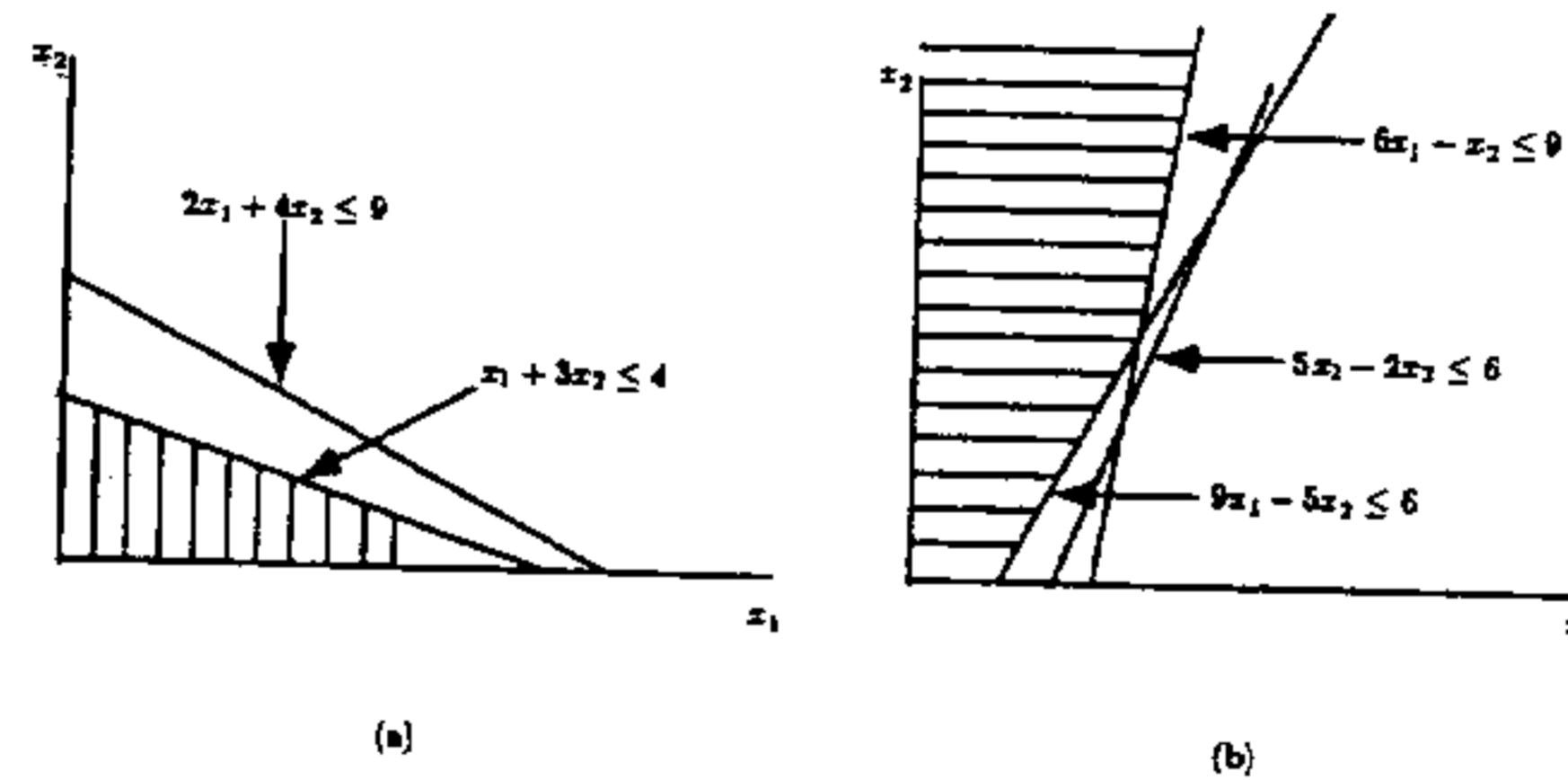


Fig. 9.1 Dominance of inequalities

again with $n = 2$, suppose that $P = \{x \in R_+^2 : 6x_1 - x_2 \leq 9, 9x_1 - 5x_2 \leq 6\}$. Now consider another valid inequality $5x_1 - 2x_2 \leq 6$ for P . Taking weights $u = (\frac{1}{3}, \frac{1}{3})$, we see that $5x_1 - 2x_2 \leq 6$ is redundant. See Figure 9.1b. ■

Where $P = \text{conv}(X)$ is not known explicitly, checking redundancy may be very difficult. Theoretically it is important to know which inequalities are needed or nonredundant in the description of P . Practically, the important point is to avoid using an inequality when one that dominates it is readily available.

In the next subsection, we discuss polyhedra and characterize which in-

9.2.2 Polyhedra, Faces, and Facets

The goal here is to understand which are the important inequalities that are necessary in describing a polyhedron, and hence at least in theory provide the best possible cuts.

For simplicity we limit the discussion to polyhedra $P \subseteq R^n$ that contain n linearly independent directions. Such polyhedra are called *full-dimensional*. Full-dimensional polyhedra have the property that there is no equation $ax = b$ satisfied at equality by all points $x \in P$.

Theorem 9.1 *If P is a full-dimensional polyhedron, it has a unique minimal description*

$$P = \{x \in R^n : a^i x \leq b_i \text{ for } i = 1, \dots, m\},$$

where each inequality is unique to within a positive multiple.

This means that if one of the inequalities in the minimal description is removed, the resulting polyhedron is no longer P , so each of the inequalities is *necessary*. On the other hand every valid inequality $\pi x \leq \pi_0$ for P that is not a positive multiple of one of the inequalities $a^i x \leq b_i$ for some i with $1 \leq i \leq m$ is redundant in the sense of Definition 9.2 as it is a nonnegative combination of two or more valid inequalities.

We now discuss another way in which the necessary inequalities can be characterized.

Definition 9.3 The points $x^1, \dots, x^k \in R^n$ are *affinely independent* if the $k-1$ directions $x^2 - x^1, \dots, x^k - x^1$ are linearly independent, or alternatively the k vectors $(x^1, 1), \dots, (x^k, 1) \in R^{n+1}$ are linearly independent.

Definition 9.4 The *dimension* of P , denoted $\dim(P)$, is one less than the maximum number of affinely independent points in P .

This means that $P \subseteq R^n$ is full-dimensional if and only if $\dim(P) = n$.

Definition 9.5 (i) F defines a *face* of the polyhedron P if $F = \{x \in P : \pi x = \pi_0\}$ for some valid inequality $\pi x \leq \pi_0$ of P .

(ii) F is a *facet* of P if F is a face of P and $\dim(F) = \dim(P) - 1$.

(iii) If F is a face of P with $F = \{x \in P : \pi x = \pi_0\}$, the valid inequality $\pi x \leq \pi_0$ is said to *represent* or *define* the face.

It follows that the faces of polyhedra are polyhedra, and it can be shown that the number of faces of a polyhedron is finite. Now we establish a way to recognize the necessary inequalities.

Proposition 9.2 *If P is full-dimensional, a valid inequality $\pi x \leq \pi_0$ is necessary in the description of P if and only if it defines a facet of P .*

So for full-dimensional polyhedra, $\pi x \leq \pi_0$ defines a facet of P if and only if there are n affinely independent points of P satisfying it at equality.

Example 9.2 Consider the polyhedron $P \subseteq R^2$, shown in Figure 9.2, described by the inequalities

$$\begin{aligned} x_1 &\leq 2 \\ x_1 + x_2 &\leq 4 \\ x_1 + 2x_2 &\leq 10 \\ x_1 + 2x_2 &\leq 6 \\ x_1 + x_2 &\geq 2 \\ x_1 &\geq 0 \\ x_2 &\geq 0. \end{aligned}$$

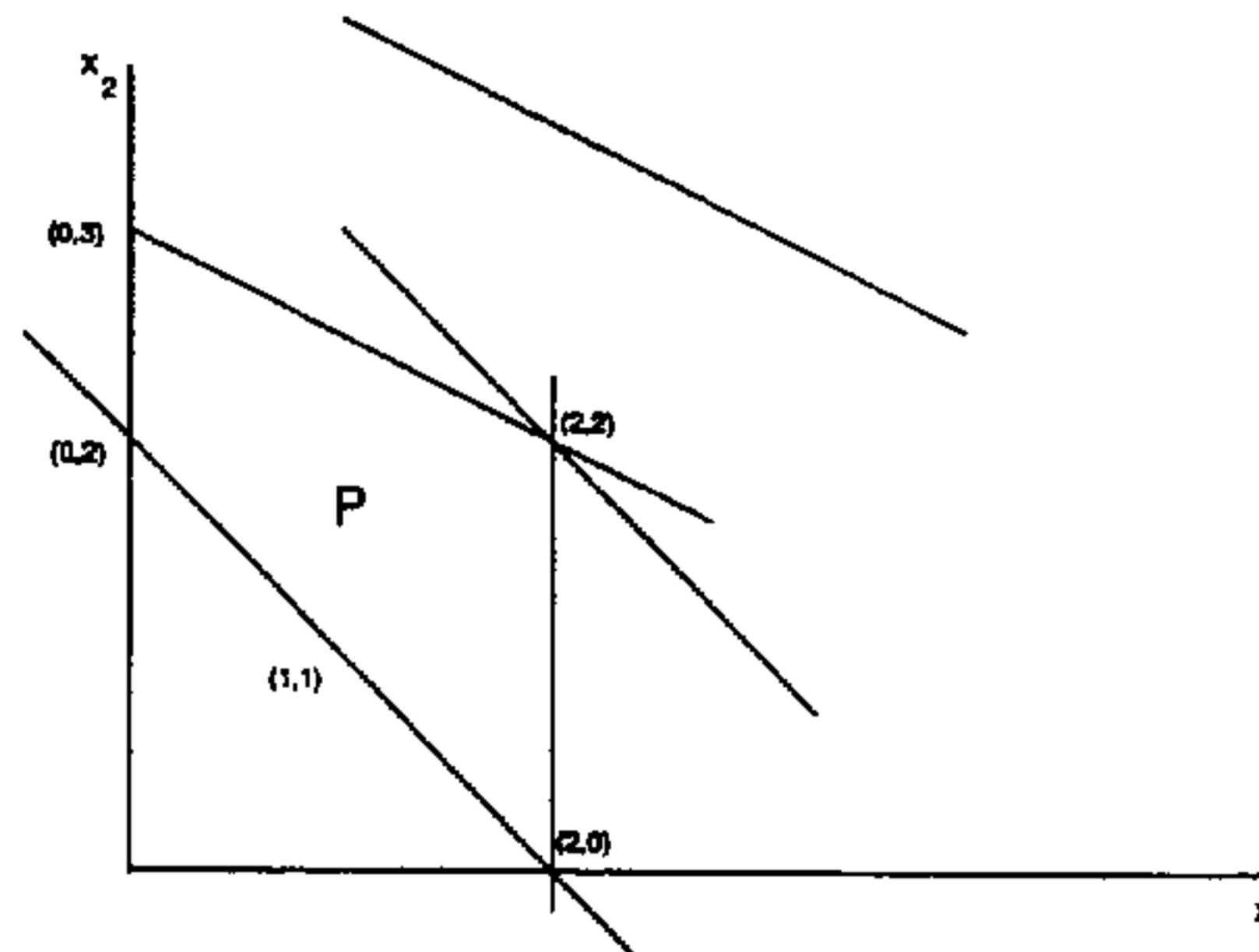


Fig. 9.2 Facets and faces of a polyhedron

P is full-dimensional as $(2, 0)$, $(1, 1)$, and $(2, 2)$ are three affinely independent points in P .

The inequality $x_1 \leq 2$ defines a facet of P as $(2, 0)$ and $(2, 2)$ are two affinely independent points in P satisfying $x_1 \leq 2$ at equality. Similarly the inequalities $x_1 + 2x_2 \leq 6$, $x_1 + x_2 \geq 2$ and $x_1 \geq 0$ define facets.

On the other hand, the inequality $x_1 + x_2 \leq 4$ defines a face consisting of just one point $(2, 2)$ of P , and hence it is redundant. Alternatively, considering the inequalities $x_1 \leq 2$ and $x_1 + 2x_2 \leq 6$ with weights $u = (\frac{1}{2}, \frac{1}{2})$ also shows that $x_1 + x_2 \leq 4$ is redundant.

The inequality $x_2 \geq 0$ is the sum of the inequalities $x_1 \leq 2$ and $-x_1 - x_2 \leq -2$ and so it is also redundant.

The minimal description is given by

$$\begin{aligned} x_1 &\leq 2 \\ x_1 + 2x_2 &\leq 6 \\ x_1 + x_2 &\geq 2 \\ x_2 &\geq 0. \end{aligned}$$

9.2.3 Facet and Convex Hull Proofs*

This subsection is for those interested in proving results about the strength of certain inequalities or formulations. The aim is to indicate ways to show that a valid inequality is facet-defining, or that a set of inequalities describes the convex hull of some discrete set $X \subset Z_+^n$.

For simplicity we assume throughout this subsection that $\text{conv}(X)$ is bounded as well as full-dimensional. So there are no hyperplanes containing all the points of X . As example we take the set $X = \{(x, y) \in R_+^m \times B^1 : \sum_{i=1}^m x_i \leq my, x_i \leq 1 \text{ for } i = 1, \dots, m\}$ that arises in Sections 1.6 and 8.4 in formulating the uncapacitated facility location problem.

Problem 1. Given $X \subset Z_+^n$ and a valid inequality $\pi x \leq \pi_0$ for X , show that the inequality defines a facet of $\text{conv}(X)$.

We consider two different approaches.

Approach 1. (Just use the definition.) Find n points $x^1, \dots, x^n \in X$ satisfying $\pi x = \pi_0$, and then prove that these n points are affinely independent.

Approach 2. (An indirect but useful way to verify the affine independence.)

(i) Select $t \geq n$ points $x^1, \dots, x^t \in X$ satisfying $\pi x = \pi_0$. Suppose that all these points lie on a generic hyperplane $\mu x = \mu_0$.

(ii) Solve the linear equation system

$$\sum_{j=1}^n \mu_j x_j^k = \mu_0 \text{ for } k = 1, \dots, t$$

in the $n + 1$ unknowns (μ, μ_0) .

(iii) If the only solution is $(\mu, \mu_0) = \lambda(\pi, \pi_0)$ for $\lambda \neq 0$, then the inequality $\pi x \leq \pi_0$ is facet-defining.

Example 9.3 Taking $X = \{(x, y) \in R_+^m \times B^1 : \sum_{i=1}^m x_i \leq my, x_i \leq 1 \text{ for } i = 1, \dots, m\}$, we have that $\dim(\text{conv}(X)) = m + 1$. Now we consider the valid inequality $x_i \leq y$ and show that it is facet-defining using Approach 2.

We select the simplest points $(0, 0)$, $(e_i, 1)$ and $(e_i + e_j, 1)$ for $j \neq i$ that are feasible and satisfy $x_i = y$.

As $(0, 0)$ lies on $\sum_{k=1}^m \mu_k x_k + \mu_{m+1} y = \mu_0$, $\mu_0 = 0$.

As $(e_i, 1)$ lies on the hyperplane $\sum_{k=1}^m \mu_k x_k + \mu_{m+1} y = 0$, $\mu_i = -\mu_{m+1}$.

As $(e_i + e_j, 1)$ lies on the hyperplane $\sum_{k=1}^m \mu_k x_k - \mu_i y = 0$, $\mu_j = 0$ for $j \neq i$.

So the hyperplane is $\mu_i x_i - \mu_i y = 0$, and $x_i \leq y$ is facet-defining. ■

Problem 2. Show that the polyhedron $P = \{x \in R^n : Ax \leq b\}$ describes $\text{conv}(X)$.

Here we present eight approaches.

Approach 1. Show that the matrix A , or the pair (A, b) have special structure guaranteeing that $P = \text{conv}(X)$.

Example 9.4 Take $X = \{(x, y) \in R_+^m \times B^1 : \sum_{i=1}^m x_i \leq my, x_i \leq 1 \text{ for } i = 1, \dots, m\}$, and consider the polyhedron/formulation

$$P = \{(x, y) \in R_+^m \times R^1 : x_i \leq y \text{ for } i = 1, \dots, m, y \leq 1\}.$$

Observe that the constraints $x_i - y \leq 0$ for $i = 1, \dots, m$ lead to a matrix with a coefficient of +1 and -1 in each row. Such a matrix is TU; see Proposition 3.2. Adding the bound constraints still leaves a TU matrix. Now as the requirements vector is integer, it follows from Proposition 3.3 that all basic solutions are integral, and $P = \text{conv}(X)$. ■

Approach 2. Show that points $(x, y) \in P$ with y fractional are not extreme points of P .

Example 9.4 (cont) Suppose that $(x^*, y^*) \in P$ with $0 < y^* < 1$. Note first that $(0, 0) \in P$. Also as $x_i^* \leq y^*$, the point $(\frac{x_1^*}{y^*}, \dots, \frac{x_m^*}{y^*}, 1) \in P$. But now

$$(x^*, y^*) = (1 - y^*)(0, 0) + y^*(\frac{x_1^*}{y^*}, \dots, \frac{x_m^*}{y^*}, 1)$$

is a convex combination of two points of P and is not extreme. Thus all vertices of P have y^* integer. ■

Approach 3. Show that for all $c \in R^n$, the linear program $z^{LP} = \max\{cx : Ax \leq b\}$ has an optimal solution $x^* \in X$.

Example 9.4 (cont) Consider the linear program $z^{LP} = \max\{\sum_{i=1}^m c_i x_i + fy : 0 \leq x_i \leq y \text{ for } i = 1, \dots, m, y \leq 1\}$. Consider an optimal solution (x^*, y^*) . Because of the constraints $0 \leq x_i \leq y$, any optimal solution has $x_i^* = y^*$ if $c_i > 0$ and $x_i^* = 0$ if $c_i < 0$. The corresponding solution value is $(\sum_{i=1}^m \max\{0, c_i\} + f)y^*$ if $y^* > 0$ and 0 otherwise. Obviously if $(\sum_{i=1}^m \max\{0, c_i\} + f) > 0$, the objective is maximized by setting $y^* = 1$, and otherwise $y^* = 0$ is optimal. Thus there is always an optimal solution with y integer, and $z^{LP} = \max\{c^T x : x \in X\}$. ■

Approach 4. Show that for all $c \in R^n$, there exists a point $x^* \in X$ and a feasible solution u^* of the dual LP $w^{LP} = \min\{ub, uA = c, u \geq 0\}$ with $cx^* = u^*b$. Note that this implies that the condition of Approach 3 is satisfied.

Example 9.4 (cont) The dual linear program is

$$\begin{aligned} \min t \\ w_i \geq c_i \text{ for } i = 1, \dots, m \\ -\sum_{i=1}^m w_i + t \geq f \\ w_i \geq 0 \text{ for } i = 1, \dots, m, t \geq 0. \end{aligned}$$

Consider the two points $(0,0)$ and $(x^*, 1)$ with $x_i^* = 1$ if $c_i > 0$ and $x_i^* = 0$ otherwise. Taking the better of the two leads to a primal solution of value $(\sum_{i:c_i > 0} c_i + f)^+$. The point $w_i = c_i^+$ for $i = 1, \dots, m$ and $t = (\sum_{i:c_i > 0} c_i + f)^+$ is clearly feasible in the dual. Thus we have found a point in X and a dual solution of the same value.

Approach 5. Show that if $\pi x \leq \pi_0$ defines a facet of $\text{conv}(X)$, then it must be identical to one of the inequalities $a^i x \leq b_i$ defining P .

Example 9.4 (cont) Consider the inequality $\sum_{i=1}^m \pi_i x_i + \pi_{m+1} y \leq \pi_0$. Let $S = \{i \in \{1, \dots, m\} : \pi_i > 0\}$ and $T = \{i \in \{1, \dots, m\} : \pi_i < 0\}$. Note that as the point $(0,0) \in X$, $\pi_0 \geq 0$, and as $(e^S, 1) \in X$, $\sum_{i \in S} \pi_i + \pi_{m+1} \leq \pi_0$, where e^S is the characteristic vector of S . Also a facet-defining inequality must have a tight point with $y = 1$. The point $(e^S, 1)$ maximizes the lhs, and so $\sum_{i \in S} \pi_i + \pi_{m+1} = \pi_0 \geq 0$.

Now consider the valid inequality obtained as a nonnegative combination of valid inequalities:

$$\begin{aligned} x_i - y \leq 0 \text{ with weight } \pi_i \text{ for } i \in S \\ -x_i \leq 0 \text{ with weight } -\pi_i \text{ for } i \in T \\ y \leq 1 \text{ with weight } \sum_{i \in S} \pi_i + \pi_{m+1}. \end{aligned}$$

The resulting inequality is $\sum_{i=1}^m \pi_i x_i + \pi_{m+1} y \leq \sum_{i \in S} \pi_i + \pi_{m+1}$. This dominates or equals the original inequality as $\sum_{i \in S} \pi_i + \pi_{m+1} \leq \pi_0$. So the only inequalities that are not nonnegative combinations of other inequalities are those describing P .

Approach 6. Show that for any $c \in R^n, c \neq 0$, the set of optimal solutions $M(c)$ to the problem $\max\{cx : x \in X\}$ lies in $\{x : a^i x = b_i\}$ for some $i = 1, \dots, m$, where $a^i x \leq b_i$ for $i = 1, \dots, m$ are the inequalities defining P .

Example 9.4 (cont) Consider an arbitrary objective $(c, f) \in R^m \times R^1$.

- If $f > 0$, $y = 1$ in every optimal solution and so $M(c, f) = \{(x, y) : y = 1\}$
- If $c_i < 0$, then $x_i = 0$ in every optimal solution
- If $c_i > 0$ and $f = 0$, then $x_i = y$ in every optimal solution
- If $c_i = 0$ for all i and $f = 0$, then $x_i = 0$ in every optimal solution.

All cases have been covered, and so $P = \text{conv}(X)$.

Approach 7. Verify that $b \in Z^n$, and show that for all $c \in Z^n$, the optimal value of the dual w^{LP} is integer valued. This is to show that the inequalities $a^i x \leq b$ form a TDI system, see Theorem 3.14.

Example 9.4 (cont) We have shown using Approach 4 that $w^{LP} = (\sum_{i:c_i > 0} c_i + f)^+$. This is integer valued when c and f are integral.

Approach 8. (Projection from an Extended Formulation). Suppose $Q \subseteq R^n \times R^p$ is a polyhedron with $P = \text{proj}_x(Q)$ as defined in Section 1.7. Show that for all $c \in R^n$, the linear program $\max\{cx : (x, w) \in Q\}$ has an optimal solution with $x \in X$.

Example 9.5 (Uncapacitated Lot-Sizing). It can be shown that solving the extended formulation presented in Section 1.6 as a linear program gives a solution with the set-up variables y_1, \dots, y_n integral, and thus provides an optimal solution to *ULS*. So its projection to the (x, y, s) space describes the convex hull of solutions to *ULS*.

9.3 0-1 KNAPSACK INEQUALITIES

Consider the set $X = \{x \in B^n : \sum_{j=1}^n a_j x_j \leq b\}$. Complementing variables if necessary by setting $\bar{x}_j = 1 - x_j$, we assume throughout this section that the coefficients $\{a_j\}_{j=1}^n$ are positive. Also we assume $b > 0$. Let $N = \{1, \dots, n\}$.

9.3.1 Cover Inequalities

Definition 9.6 A set $C \subseteq N$ is a *cover* if $\sum_{j \in C} a_j > b$. A cover is *minimal* if $C \setminus \{j\}$ is not a cover for any $j \in C$.

Note that C is a cover if and only if its associated incidence vector x^C is infeasible for X .

Proposition 9.3 If $C \subseteq N$ is a cover for X , the cover inequality

$$\sum_{j \in C} x_j \leq |C| - 1$$

is valid for X .

Proof. We show that if x^R does not satisfy the inequality, then $x^R \notin X$. If $\sum_{j \in C} x_j^R > |C| - 1$, then $|R \cap C| = |C|$ and thus $R \supseteq C$. Then $\sum_{j \in R} a_j x_j^R = \sum_{j \in C} a_j x_j^R + \sum_{j \in R \setminus C} a_j x_j^R > b$ and so $x^R \notin X$.

$$X = \{x \in B^7 : 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19\}.$$

Some minimal cover inequalities for X are:

$$\begin{array}{rcccccccl} x_1 & +x_2 & +x_3 & & & & & \leq & 2 \\ x_1 & +x_2 & & & +x_6 & & & \leq & 2 \\ x_1 & & & & +x_5 & +x_6 & & \leq & 2 \\ & & x_3 & +x_4 & +x_5 & +x_6 & & \leq & 3. \end{array}$$

9.3.2 Strengthening Cover Inequalities

Are the cover inequalities "strong"? Is it possible to strengthen the cover inequalities so that they provide better cuts?

First we observe that there is a simple way to strengthen the basic cover inequality.

Proposition 9.4 *If C is a cover for X , the extended cover inequality*

$$\sum_{j \in E(C)} x_j \leq |C| - 1$$

is valid for X , where $E(C) = C \cup \{j : a_j \geq a_i \text{ for all } i \in C\}$.

The proof of validity is almost identical to that of Proposition 9.3.

Example 9.6 (cont) The extended cover inequality for $C = \{3, 4, 5, 6\}$ is $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$. So the cover inequality $x_3 + x_4 + x_5 + x_6 \leq 3$ is dominated by the valid inequality $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$.

Observe, however, that this is in turn dominated by the inequality $2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$.

Can we define a procedure that allows us to find the last inequality, which is nonredundant (facet-defining) and thus as strong as possible?

Example 9.6 (cont) Consider again the cover inequality for $C = \{3, 4, 5, 6\}$. Clearly when $x_1 = x_2 = x_7 = 0$, the inequality $x_3 + x_4 + x_5 + x_6 \leq 3$ is valid for $\{x \in B^4 : 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19\}$.

Now keeping $x_2 = x_7 = 0$, we ask for what values of α_1 , the inequality

$$\alpha_1 x_1 + x_3 + x_4 + x_5 + x_6 \leq 3$$

is valid for $\{x \in B^5 : 11x_1 + 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19\}$?

When $x_1 = 0$, the inequality is known to be valid for all values of α_1 .

When $x_1 = 1$, it is a valid inequality

$6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19 - 11$, or equivalently

if and only if $\alpha_1 + \max\{x_3 + x_4 + x_5 + x_6 : 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 8, x \in B^4\} \leq 3$,

or equivalently

if and only if $\alpha_1 \leq 3 - \zeta$, where $\zeta = \max\{x_3 + x_4 + x_5 + x_6 : 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 8, x \in B^4\}$.

Now $\zeta = 1$ at the point $x = (0, 0, 0, 1)$, and hence $\alpha_1 \leq 2$.

Thus the inequality is valid for all values of $\alpha_1 \leq 2$, and $\alpha_1 = 2$ gives the strongest inequality. ■

In general the problem is to find best possible values for α_j for $j \in N \setminus C$ such that the inequality

$$\sum_{j \in C} x_j + \sum_{j \in N \setminus C} \alpha_j x_j \leq |C| - 1$$

is valid for X .

The procedure we now describe leads to such a set of values and in fact provides a facet-defining inequality for $\text{conv}(X)$ when C is a minimal cover and $a_j \leq b$ for all $j \in N$.

Procedure to Lift Cover Inequalities

Let j_1, \dots, j_r be an ordering of $N \setminus C$. Set $t = 1$.

The valid inequality $\sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in C} x_j \leq |C| - 1$ has been obtained so far. To calculate the largest value of α_{j_t} for which the inequality $\alpha_{j_t} x_{j_t} + \sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in C} x_j \leq |C| - 1$ is valid, solve the knapsack problem:

$$\begin{aligned} \zeta_t &= \max \sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in C} x_j \\ &\sum_{i=1}^{t-1} a_{j_i} x_{j_i} + \sum_{j \in C} a_j x_j \leq b - a_{j_t} \\ &x \in \{0, 1\}^{|C|+t-1}. \end{aligned}$$

Set $\alpha_{j_t} = |C| - 1 - \zeta_t$.

Stop if $t = r$.

It can be seen that ζ_t measures how much space is used up by the variables $\{j_1, \dots, j_{t-1}\} \cup C$ in the lifted inequality when $x_{j_t} = 1$.

Example 9.6 (cont) If we take $C = \{3, 4, 5, 6\}$, $j_1 = 1$, $j_2 = 2$ and $j_3 = 7$, we have already been through the steps to calculate $\alpha_1 = 2$. Continuing with

$\max\{2x_1+x_3+x_4+x_5+x_6 : 11x_1+6x_3+5x_4+5x_5+4x_6 \leq 19-6 = 13, x \in B^6\}$
 $= 2$, and thus $\alpha_{j_2} = \alpha_2 = 3 - 2 = 1$.

A similar calculation shows that $\zeta_3 = 3$ and thus $\alpha_{j_3} = \alpha_7 = 0$. Thus we finish with a facet-defining inequality $2x_1+1x_2+1x_3+1x_4+1x_5+1x_6+0x_7 \leq 3$.

Returning to the question of the strength of the cover inequalities, it is not difficult to show that $\sum_{j \in C} x_j \leq |C| - 1$ is facet-defining for $\text{conv}(X')$ where $X' = \{x \in B^{|C|} : \sum_{j \in C} a_j x_j \leq b\}$. This suggests that the inequality is strong. On the other hand, it is clearly strengthened by the lifting procedure that terminates with a facet-defining inequality for $\text{conv}(X)$.

9.3.3 Separation for Cover Inequalities

Now let \mathcal{F} be the family of cover inequalities for X , and let us examine the separation problem for this family. Explicitly we are given a nonintegral point x^* with $0 \leq x_j^* \leq 1$ for all $j \in N$, and we wish to know whether x^* satisfies all the cover inequalities. To formalize this problem, note that the cover inequality can be rewritten as

$$\sum_{j \in C} (1 - x_j) \geq 1.$$

Thus it suffices to answer the question:

Does there exist a set $C \subseteq N$ with $\sum_{j \in C} a_j > b$ for which $\sum_{j \in C} (1 - x_j^*) < 1$?, or put slightly differently,

Is $\zeta = \min_{C \subseteq N} \{\sum_{j \in C} (1 - x_j^*) : \sum_{j \in C} a_j > b\} < 1$?

As the set C is unknown, this can be formulated as a 0-1 integer program where the variable $z_j = 1$ if $j \in C$ and $z_j = 0$ otherwise. So the question can be restated yet again,

Is $\zeta = \min \{\sum_{j \in N} (1 - x_j^*) z_j : \sum_{j \in N} a_j z_j > b, z \in B^n\} < 1$?

Theorem 9.5 (i) If $\zeta \geq 1$, x^* satisfies all the cover inequalities.

(ii) If $\zeta < 1$ with optimal solution z^R , the cover inequality $\sum_{j \in R} x_j \leq |R| - 1$ cuts off x^* by an amount $1 - \zeta$.

Example 9.7 Consider the 0-1 knapsack set

$$X = \{x \in B^6 : 45x_1 + 46x_2 + 79x_3 + 54x_4 + 53x_5 + 125x_6 \leq 178\}.$$

The cover separation problem is:

$$\begin{aligned} \min & 1z_1 + 1z_2 + \frac{1}{4}z_3 + \frac{1}{2}z_4 + 0z_5 + 1z_6 \\ & 45z_1 + 46z_2 + 79z_3 + 54z_4 + 53z_5 + 125z_6 > 178 \\ & z \in B^6. \end{aligned}$$

An optimal solution is $z^R = (0, 0, 1, 1, 1, 0)$ with $\zeta = \frac{3}{4}$. Thus the inequality

$$x_3 + x_4 + x_5 \leq 2$$

is violated by $1 - \zeta = \frac{1}{4}$.

9.4 MIXED 0-1 INEQUALITIES

Here we consider the mixed 0-1 set:

$$X = \{(x, y) \in R_+^n \times B^n : \sum_{j \in N_1} x_j \leq b + \sum_{j \in N_2} x_j, x_j \leq a_j y_j \text{ for } j \in N_1 \cup N_2\}.$$

Note that when $N_2 = \emptyset$ and $x_j = a_j y_j$ for all $j \in N_1$, this reduces to the knapsack set studied in Section 9.3. The set X can be viewed as the feasible region of a simple fixed charge flow network (see Figure 9.3).

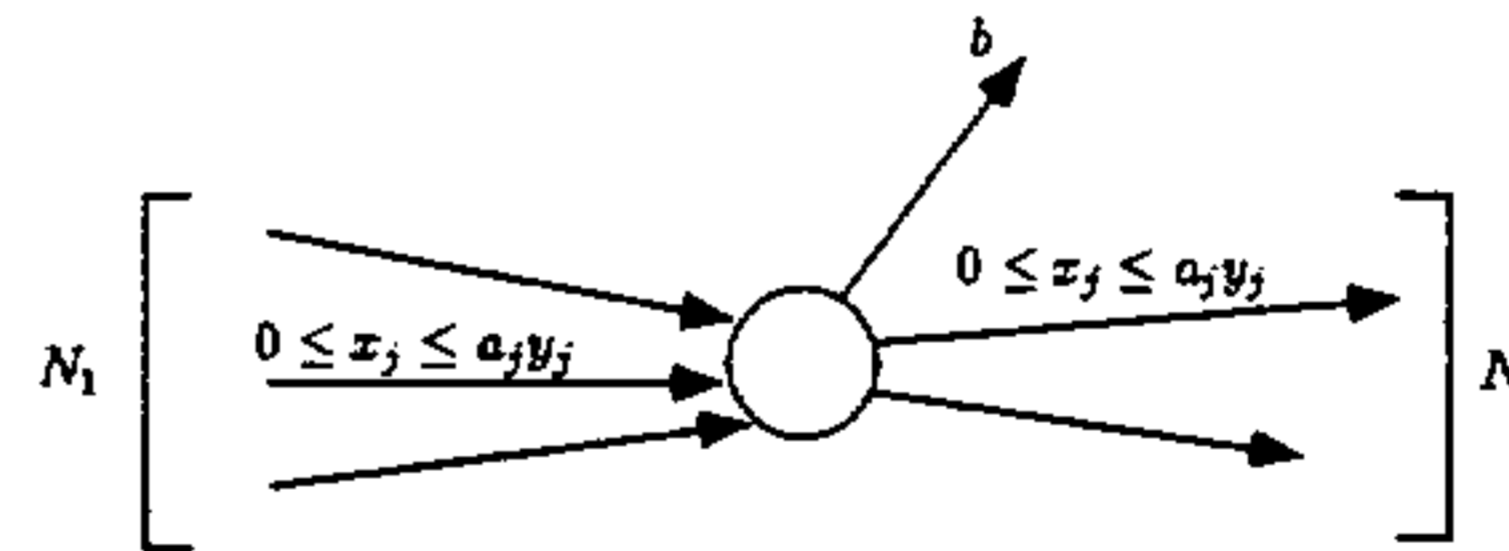


Fig. 9.3 Mixed 0-1 flow

9.4.1 Flow Cover Inequalities

Definition 9.7 A set $C = C_1 \cup C_2$ with $C_1 \subseteq N_1, C_2 \subseteq N_2$ is a *generalized cover* for X if $\sum_{j \in C_1} a_j - \sum_{j \in C_2} a_j = b + \lambda$ with $\lambda > 0$. λ is called the (cover-)excess.

Proposition 9.6 The flow cover inequality

$$\sum_{j \in C_1} x_j + \sum_{j \in C_1} (a_j - \lambda)^+(1 - y_j) \leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} y_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} x_j$$

is valid for X , where $L_2 \subseteq N_2 \setminus C_2$.

Proof. Let $C_1^+ = \{j \in C_1 : a_j > \lambda\}$. Now given a point $(x, y) \in X$, we must show that (x, y) satisfies the inequality. Let $T = \{j \in N_1 \cup N_2 : y_j = 1\}$. There are two cases.

Case 1. $|C_1^+ \setminus T| + |L_2 \cap T| = 0$. Now

$$\begin{aligned} & \sum_{j \in C_1} x_j + \sum_{j \in C_1} (a_j - \lambda)^+(1 - y_j) \\ = & \sum_{j \in C_1 \cap T} x_j + \sum_{j \in C_1^+ \setminus T} (a_j - \lambda) \\ = & \sum_{j \in C_1 \cap T} x_j \quad (\text{as } |C_1^+ \setminus T| = 0) \\ \leq & \sum_{j \in N_1} x_j \quad (\text{as } x_j \geq 0) \\ \leq & b + \sum_{j \in N_2} x_j \quad (\text{by definition of } X) \\ = & b + \sum_{j \in C_2} x_j + \sum_{j \in L_2 \cap T} x_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} x_j \\ \leq & b + \sum_{j \in C_2} a_j + 0 + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} x_j \quad (\text{as } |L_2 \cap T| = 0) \\ = & b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} y_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} x_j. \end{aligned}$$

Case 2. $|C_1^+ \setminus T| + |L_2 \cap T| \geq 1$.

$$\begin{aligned} & \sum_{j \in C_1} x_j + \sum_{j \in C_1} (a_j - \lambda)^+(1 - y_j) \\ = & \sum_{j \in C_1 \cap T} x_j + \sum_{j \in C_1^+ \setminus T} (a_j - \lambda) \\ \leq & \sum_{j \in C_1} a_j - |C_1^+ \setminus T| \lambda \quad (\text{as } x_j \leq a_j) \\ \leq & \sum_{j \in C_1} a_j - \lambda + \lambda |L_2 \cap T| \quad (\text{as } -|C_1^+ \setminus T| \leq -1 + |L_2 \cap T|) \\ = & b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} y_j \\ \leq & b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} y_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} x_j \quad (\text{as } x_j \geq 0). \end{aligned}$$

Example 9.8 Consider the set

$$X = \{(x, y) \in R_+^6 \times B^6 : x_1 + x_2 + x_3 \leq 4 + x_4 + x_5 + x_6,$$

$$x_1 \leq 3y_1, x_2 \leq 3y_2, x_3 \leq 6y_3, x_4 \leq 3y_4, x_5 \leq 5y_5, x_6 \leq 1y_6\}.$$

Taking $C_1 = \{1, 3\}$ and $C_2 = \{4\}$, $C = (C_1, C_2)$ is a generalized cover with $\lambda = 2$. Taking $L_2 = \{5\}$, the resulting flow cover inequality is

$$x_1 + x_3 + 1(1 - y_1) + 4(1 - y_3) + 7 + 2y_5 \leq 10.$$

9.4.2 Separation for Flow Cover Inequalities

If one makes the assumption that $x_j = a_j y_j$ and $a_j \geq \lambda$ for all $j \in C_1$, and that $L_2 = N_2 \setminus C_2$, the flow cover inequality becomes:

$$\sum_{j \in C_1} a_j y_j + \sum_{j \in C_1} (a_j - \lambda)(1 - y_j) \leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in N_2 \setminus C_2} y_j,$$

or after simplification,

$$\sum_{j \in C_1} a_j - \lambda \sum_{j \in C_1} (1 - y_j) \leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in N_2 \setminus C_2} y_j,$$

or dividing by λ ,

$$\sum_{j \in C_1} (1 - y_j) + \sum_{j \in N_2 \setminus C_2} y_j \geq 1,$$

or after rewriting,

$$\sum_{j \in C_1} (1 - y_j) - \sum_{j \in C_2} y_j \geq 1 - \sum_{j \in N_2} y_j.$$

This is nothing but the cover inequality for a 0-1 knapsack set with both positive and negative coefficients $\{x \in B^n : \sum_{j \in N_1} a_j y_j - \sum_{j \in N_2} a_j y_j \leq b\}$ where the cover (C_1, C_2) satisfies $\sum_{j \in C_1} a_j - \sum_{j \in C_2} a_j = b + \lambda$ with $\lambda > 0$.

This immediately suggests a separation heuristic for the flow cover inequalities. Letting z be the unknown incidence vector of $C = (C_1, C_2)$, consider the knapsack problem

$$\zeta = \min \sum_{j \in N_1} (1 - y_j^*) z_j - \sum_{j \in N_2} y_j^* z_j$$

$$\sum_{j \in N_1} a_j z_j - \sum_{j \in N_2} a_j z_j > b$$

$$z \in B^n.$$

Let z^C , the incidence vector of C , be an optimal solution.

Flow Cover Separation Heuristic. Take the cover $C = (C_1, C_2)$ obtained by solving this knapsack problem, and test whether

$$\sum_{j \in C_1} x_j^* + \sum_{j \in C_1} (a_j - \lambda)^+(1 - y_j^*) > b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} y_j^* + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} x_j^*$$

where $L_2 = \{j \in N_2 \setminus C_2 : \lambda y_j^* < x_j^*\}$. If so, a violated flow cover inequality has been found.

$(3, 0, 4, 3, 0, 0)$ and $y^* = (1, 0, \frac{1}{3}, 1, 0, 0)$. Solving the knapsack problem

$$\zeta = \min 0z_1 + 1z_2 + \frac{1}{3}z_3 - 1z_4 - 0z_5 - 0z_6$$

$$3z_1 + 3z_2 + 6z_3 - 3z_4 - 5z_5 - z_6 > 4$$

$$z \in B^6,$$

one obtains as solution $C_1 = \{1, 3\}$, $C_2 = \{4\}$, and $\lambda = 2$. The resulting flow cover inequality

$$x_1 + x_3 + 1(1 - y_1) + 4(1 - y_3) \leq 7 + x_5 + x_6$$

is violated by $\frac{4}{3}$.

As in the previous section, it is natural to ask if the flow cover inequality of Proposition 9.6 is facet-defining, and if not to attempt to strengthen the inequality by lifting. A partial answer is that the inequality is facet-defining when $x_j = y_j = 0$ for $j \in N_2$ and C_1 is a minimal cover. However calculating valid lifting coefficients is more complicated because of the presence of the $x_j \leq a_j y_j$ constraints, so it becomes a computational question whether such lifting is worthwhile in practice.

9.5 THE OPTIMAL SUBTOUR PROBLEM

Consider a variant of the traveling salesman problem in which the salesman makes a profit f_j if he visits city (client) $j \in N$, he pays travel costs c_e if he traverses edge $e \in E$, but he is not obliged to visit all the cities. His subtour must start and end at city 1, and include at least two other cities. This problem is also known as the *prize collecting traveling salesman problem*, and is a subproblem of certain vehicle routing problems. It is \mathcal{NP} -hard.

Introducing binary edge variables: $x_e = 1$ if edge e lies on the subtour and $x_e = 0$ otherwise for $e \in E$; and binary node variables: $y_j = 1$ if city j lies on the subtour and $y_j = 0$ otherwise for $j \in N$, one obtains the formulation

$$\max - \sum_{e \in E} c_e x_e + \sum_{j \in N} f_j y_j \quad (9.1)$$

$$\sum_{e \in \delta(i)} x_e = 2y_i \text{ for } i \in N \quad (9.2)$$

$$\sum_{e \in E(S)} x_e \leq \sum_{i \in S \setminus \{k\}} y_i \text{ for } k \in S, S \subseteq N \setminus \{1\} \quad (9.3)$$

$$y_1 = 1 \quad (9.4)$$

$$x \in B^{|E|}, y \in B^{|N|} \quad (9.5)$$

Constraints (9.2) ensure that if a node is visited, two adjacent edges are used to enter and leave, and (9.3) are *generalized subtour elimination constraints* (GSEC), which ensure that there is no subtour that does not contain node

consisting of node 1 and just one other node.

The question addressed here is not to find new inequalities, but how to deal with the exponential number of generalized subtour elimination constraints.

9.5.1 Separation for Generalized Subtour Constraints

Let $N' = N \setminus \{1\}$, and $E' = E \setminus \{\delta(1)\}$. To formalize the separation problem, we represent the unknown set $S \subseteq N'$ by binary variables z with $z_i = 1$ if $i \in S$. Now (x^*, y^*) violates the (k, S) generalized subtour inequality if

$$\sum_{e \in E'(S)} x_e^* > \sum_{i \in S \setminus \{k\}} y_i^*$$

if and only if $\zeta_k > 0$ where

$$\zeta_k = \max \left\{ \sum_{e=(i,j) \in E', i < j} x_e^* z_i z_j - \sum_{i \in N' \setminus \{k\}} y_i^* z_i : z \in B^{|N'|}, z_k = 1 \right\}.$$

Note that this is a quadratic 0-1 program without constraints.

We now show how this problem can be solved in polynomial time. First we convert it into a linear integer program, and then we observe that the linear programming relaxation is integral.

We introduce variables w_e with $w_e = 1$ if $z_i = z_j = 1$ for $e = (i, j) \in E'$ with $i < j$, and we obtain the integer program

$$\zeta_k = \max \sum_{e \in E'} x_e^* w_e - \sum_{i \in N' \setminus \{k\}} y_i^* z_i \quad (9.6)$$

$$(IP) \quad w_e \leq z_i, w_e \leq z_j \text{ for } e = (i, j) \in E' \quad (9.7)$$

$$w_e \geq z_i + z_j - 1 \text{ for } e = (i, j) \in E' \quad (9.8)$$

$$w \in B^{|E'|}, z \in B^{|N'|}, z_k = 1. \quad (9.9)$$

Proposition 9.7 *If $x_e^* \geq 0$ for $e \in E'$, the linear program consisting of (9.6), (9.7) and the bound constraints obtained by relaxing integrality in (9.9) always has an integer optimal solution solving IP.*

Proof. Consider the linear programming relaxation of IP. As $x_e^* \geq 0$ for all $e \in E'$, there exists an optimal solution with w_e as large as possible. Thus by (9.7), $w_e = \min(z_i, z_j)$. As $z_i \leq 1$ for all $i \in N'$, $\min(z_i, z_j) \geq z_i + z_j - 1$, and so (9.8) is automatically satisfied, and can be dropped. Now each constraint of the linear programming relaxation contains two nonzero coefficients of -1 and $+1$, and is thus totally unimodular by Proposition 3.2 ■

The constraint matrix in the dual of the linear program in Proposition 9.7 is a node-arc incidence matrix, and so the problem can be solved as a max flow problem. As $k \in \{2, \dots, n\}$, the separation problem for GSECs can be

solved exactly by solving $n - 1$ max flow problems. In practice, very fast separation heuristics have also been developed in computational studies of the TSP, which can be used to find many violated GSECs.

Example 9.9 We consider an instance of the optimal subtour problem with

$$n = 7, f = (2, 4, 1, 3, 7, 1, 7) \text{ and } (c_e) = \begin{pmatrix} - & 4 & 3 & 3 & 5 & 2 & 5 \\ - & - & 5 & 3 & 3 & 4 & 7 \\ - & - & - & 4 & 6 & 0 & 4 \\ - & - & - & - & 4 & 4 & 6 \\ - & - & - & - & - & 5 & 8 \\ - & - & - & - & - & - & 3 \\ - & - & - & - & - & - & - \end{pmatrix}$$

Iteration 0. We solve a linear programming relaxation of problem (9.1)–(9.5) consisting of the degree constraints (9.2), constraint $y_1 = 1$, the trivial GSECs $x_e \leq y_i, x_e \leq y_j$ for $e = (i, j) \in E$, and the bounds on the variables.

The resulting solution consists of two subtours (1, 5, 2, 4) and (3, 6, 7) with $z^{LP} = 4$.

Iteration 1. The constraint $x_{36} + x_{37} + x_{67} \leq y_3 + y_7$ is added cutting off the subtour (3, 6, 7).

The solution of the new relaxation consists of two subtours (1, 3, 6, 7) and (2, 4, 5) with $z^{LP} = 4$.

Iteration 2. The constraint $x_{24} + x_{25} + x_{45} \leq y_4 + y_5$ is added cutting off the subtour (2, 4, 5).

The new LP solution is shown in Figure 9.4 with $z^{LP} = 4$.

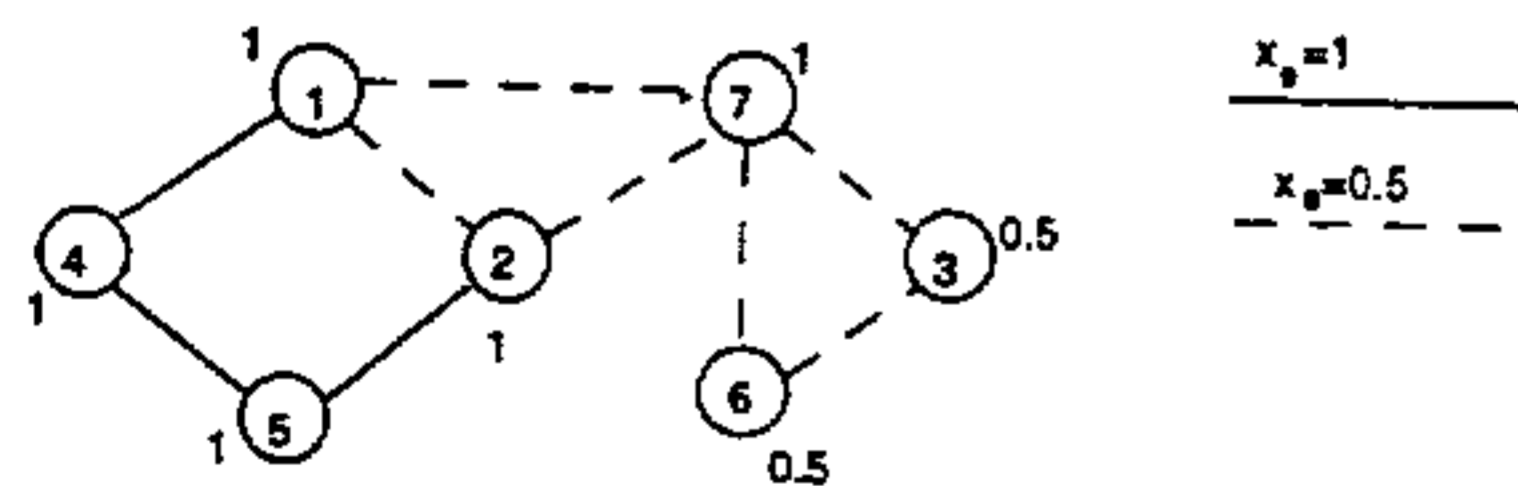


Fig. 9.4 Fractional optimal subtour solution

The separation problem for $k = 7$ is the linear program:

$$\zeta_7 = \max 1w_{45} + 1w_{25} + 0.5w_{27} + 0.5w_{67} + 0.5w_{37} + 0.5w_{36} - 1z_2 - 0.5z_3 - 1z_4 - 1z_5 - 0.5z_6$$

subject to $0 \leq w_e \leq z_i, z_j \leq 1$ for $e = (i, j) \in E', z_7 = 1$

An optimal solution is $z_3 = z_6 = z_7 = w_{45} = w_{25} = w_{67} = 1$ with violation $\zeta_7 = \frac{1}{2}$.

Iteration 3. The constraint $x_{36} + x_{37} + x_{67} \leq y_3 + y_6$ is added.

The new linear programming solution is the single subtour (1, 2, 5, 4, 7, 6), which is thus optimal with value of 2. ■

9.6 BRANCH-AND-CUT

Successful solution of difficult IPs requires a combination of the many different ideas developed in this and earlier chapters. In Chapter 7, in discussing branch-and-bound, we mentioned the importance of efficient preprocessing, fast solution and reoptimization of linear programs, and good search strategies.

The need for good primal heuristics has been stressed, and ideas for using the linear programming solution to try to construct feasible solutions are discussed in Chapter 12. In this and the previous chapter we have also considered the use of separation algorithms to generate cutting planes which tighten up the formulations.

A *Branch-and-Cut Algorithm* is a branch-and-bound algorithm in which cutting planes are generated throughout the branch-and-bound tree. Though this may seem to be a minor difference, in practice there is a change of philosophy. Rather than reoptimizing fast at each node, the new philosophy is to do as much work as is necessary to get a tight dual bound at the node. Now the goal is not only to reduce the number of nodes in the tree significantly by using cuts and improved formulations, but also by trying anything else that may be useful such as preprocessing at each node, a primal heuristic at each node, and so forth.

In practice there is obviously a trade-off. If many cuts are added at each node, reoptimization may be much slower than before. In addition, keeping all the information in the tree is significantly more difficult. In branch-and-bound the problem to be solved at each node is obtained just by adding bounds. In branch-and-cut a *cut pool* is used in which all the cuts are stored. In addition to keeping the bounds and a good basis in the node list, it is also necessary to indicate which constraints are needed to reconstruct the formulation at the given node, so pointers to the appropriate constraints in the cut pool are kept.

A flowchart of the basic steps of a branch-and-cut algorithm is shown in Figure 9.5.

Example 9.10 Generalized Assignment Problem. We consider an instance of the problem

$$\begin{aligned} \max & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} \leq 1 \text{ for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \text{ for } j = 1, \dots, n \\ & x \in B^m \end{aligned}$$

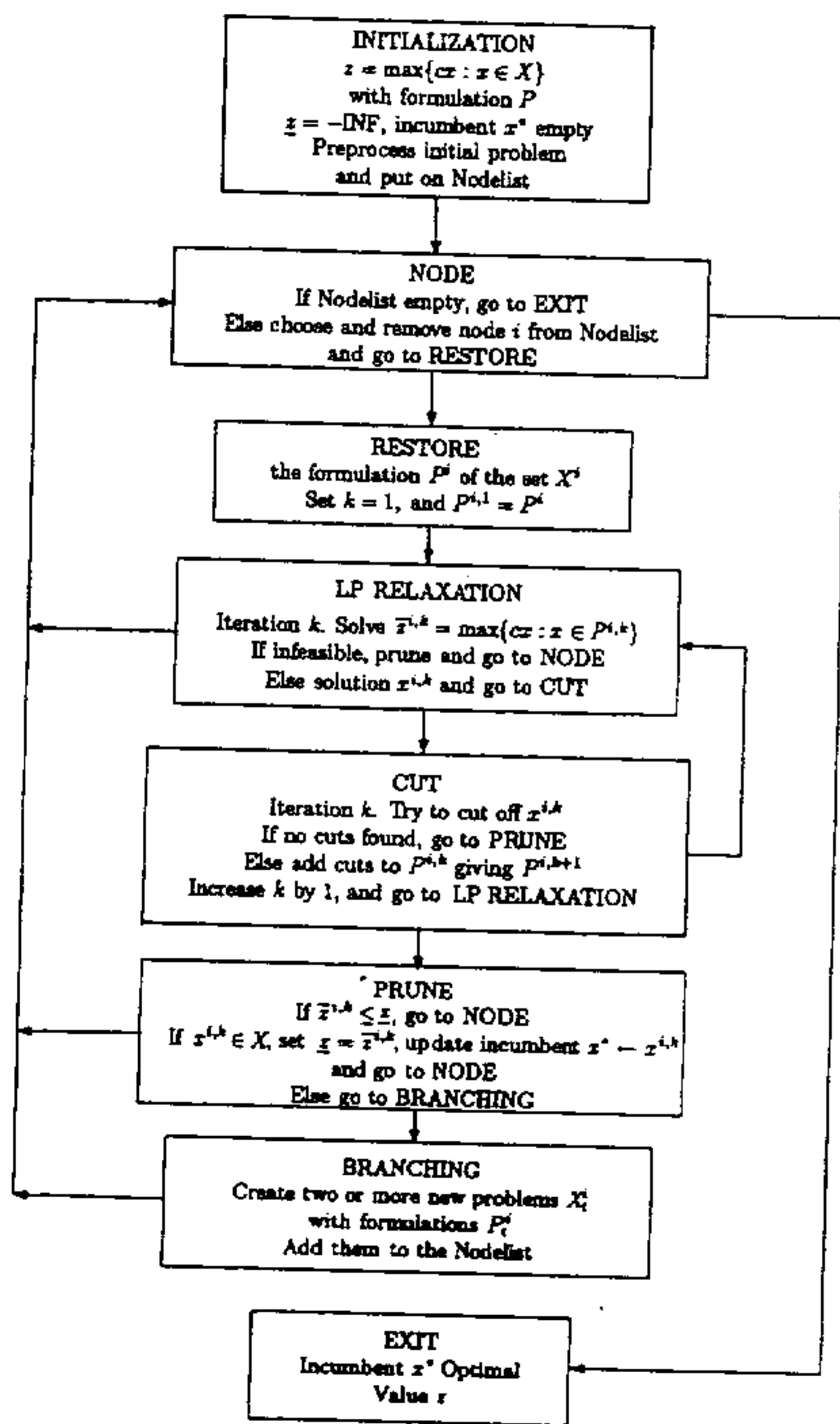


Fig. 9.5 Branch-and-cut

with $m = 10, n = 5, b = (91, 87, 109, 88, 64)^T$,

$$(c_{ij}) = \begin{pmatrix} 110 & 16 & 25 & 78 & 59 \\ 65 & 69 & 54 & 28 & 71 \\ 19 & 93 & 45 & 45 & 9 \\ 89 & 31 & 72 & 83 & 20 \\ 62 & 17 & 77 & 18 & 39 \\ 37 & 115 & 87 & 59 & 97 \\ 89 & 102 & 98 & 74 & 61 \\ 78 & 96 & 87 & 55 & 77 \\ 74 & 27 & 99 & 91 & 5 \\ 88 & 97 & 99 & 99 & 51 \end{pmatrix}, (a_{ij}) = \begin{pmatrix} 95 & 1 & 21 & 66 & 59 \\ 54 & 53 & 44 & 26 & 60 \\ 3 & 91 & 43 & 42 & 5 \\ 72 & 30 & 56 & 72 & 9 \\ 44 & 1 & 71 & 13 & 27 \\ 20 & 99 & 87 & 52 & 85 \\ 72 & 96 & 97 & 73 & 49 \\ 75 & 82 & 83 & 44 & 59 \\ 68 & 8 & 87 & 74 & 4 \\ 69 & 83 & 98 & 88 & 45 \end{pmatrix}$$

We use a branch-and-cut system generating lifted cover inequalities. Three passes are made at the top node with up to 5 cuts added per pass. One pass is made at all other tree nodes with up to 5 cuts added per pass. The resulting branch-and-cut tree is shown in Figure 9.6.

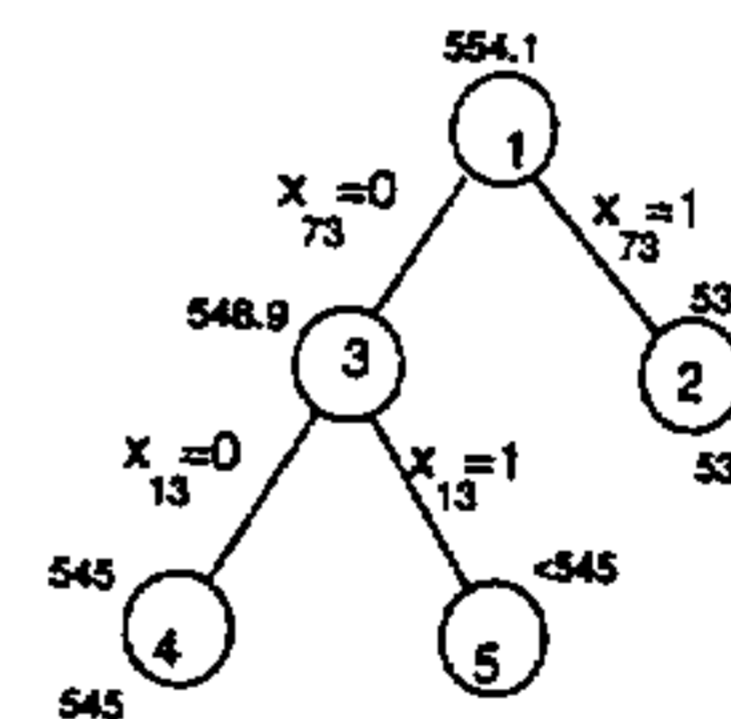


Fig. 9.6 Branch-and-cut tree

Initialization. The preprocessor eliminates the five variables $x_{11}, x_{32}, x_{62}, x_{72}, x_{65}$ with $a_{ij} > b_j$.

Node 1. The initial linear programming value is 595.6. After three passes, 14 cuts have been generated, and the improved linear programming value is 554.1, with corresponding linear programming solution: $x_{12} = 1, x_{23} = x_{25} = 0.5, x_{31} = 0.76, x_{33} = 0.24, x_{43} = x_{45} = 0.5, x_{52} = 0.74, x_{54} = 0.26, x_{61} = 1, x_{73} = x_{75} = 0.5, x_{82} = 1, x_{91} = 0.76, x_{94} = 0.24, x_{10,1} = 0.24, x_{10,4} = 0.76, x_{ij} = 0$ otherwise.

x_{73} is chosen as branching variable, creating node 2 with $x_{73} = 1$ and node 3 with $x_{73} = 0$.

Node 2. Three cuts are added. The resulting linear programming solution is integral with value 537. The incumbent value z and solution x^* are updated. The node is pruned by optimality.

Node 3. One cut is generated. The resulting linear programming solution with value 548.9 is nonintegral. x_{13} is chosen as branching variable, creating node 4 with $x_{13} = 0$ and node 5 with $x_{13} = 1$.

Node 4. The linear programming solution is integral with value 545. The incumbent value and solution are updated. The node is pruned by optimality.

Node 5. The linear programming solution value does not exceed the incumbent value of 545. The node is pruned by bound.

Termination. The node list is empty. $z = 545$ with optimal solution $x_{12} = x_{23} = x_{31} = x_{43} = x_{52} = x_{61} = x_{75} = x_{82} = x_{91} = x_{10,4} = 1$ and $x_{ij} = 0$ otherwise.

If the number of cut passes is unlimited at the top node, 18 cuts are generated, the final linear programming value is 546, and the branch-and-cut tree has just three nodes. Using just branch-and-bound, 4206 nodes are required. ■

9.7 NOTES

The use of cutting planes as a practical tool for solving general integer programs was almost completely abandoned in the 1960s and 1970s. However, the interest in looking for strong valid inequalities for \mathcal{NP} -hard problems in the early 1970s can perhaps be traced to two sources:

- (i) The beautiful polyhedral results of Edmonds and Fulkerson in the late 1960s in deriving the convex hulls of the sets of incidence vectors for matching, branching, and matroid problems, as well as the results of Gomory, and Gomory and Johnson, in developing characterizations of the valid inequalities for group and integer programming problems, and
- (ii) The idea that because of \mathcal{NP} -completeness theory, there is no hope of an efficient algorithm for integer programming, and therefore that it is important to study specific problem structure if progress on \mathcal{NP} -hard problems is to be made.

The first polyhedral studies in the early seventies, other than those of the knapsack polytope, were of the stable set polytope [Pad73], [NemTro74] and the TSP polytope [GroPad75].

9.2 Books on polyhedra include [Gru67], [StoWit70], and [Zie95]. Surveys specially written with integer programming and combinatorial optimization in view are [Pul83] and Chapter I.1.4 in [NemWol88].

9.3 Cover inequalities for 0-1 knapsack polytopes were developed simultaneously in [Balas75b], [HamJohPel75], and [Wol75a]. The concept of lifting, already present in [Gom69], was extended in [Pad73], [Wol76b]. A major step

forward came with the use of these inequalities in a cutting plane/branch-and-bound algorithm for 0-1 IPs [CroJohPad83]. Twenty years later separation routines for these inequalities are finally making their way into the commercial codes such as CPLEX, XOSL, and XPRESS.

Recent developments include the derivation of larger classes of inequalities [Wei97] for 0-1 knapsacks, and the generalization of cover inequalities to 0-1 knapsacks with GUB constraints [Wol90], [Gueta195], and to integer knapsacks [Cerietal95].

9.4 Flow cover inequalities appeared first in [PadVanRWol85] and were implemented computationally in [VRoWol87]. Other related inequalities for the model appear in [Sta92]. Recently stronger liftings of the flow cover inequalities and extensive computational tests are presented in [Gueta196]. A new mixed integer model consisting of a 0-1 knapsack constraint plus a single continuous variable is studied in [MarWol97].

9.5 The optimal subtour problem is studied in [Balas89]; see also [Bau97]. The GSEC inequalities are proposed by [Goe94] in studying the Steiner tree problem. The reduction of the quadratic 0-1 problem to the dual of a network flow problem is folklore. The reduction to a series of max flow problems is in [Rhy70]. The problem of finding violated subtour elimination constraints quickly is faced by all the researchers developing computational studies of the TSP problem. [PadRin91] present very effective heuristic separation routines. A recent survey on TSP is [JueReiRin95].

9.6 Two recent surveys on branch-and-cut are [JueReiThi95] and [LucBea96]. See also the annotated bibliography [CapFis97] which contains references to applications in general mixed integer programming, routing, scheduling, graph partitioning, set packing and covering, network design, and location problems among others.

Three research codes especially built to allow the easy development of branch-and-cut applications are MINTO [NemSavSig94], ABACUS [Thi95] and *bc-opt* [Cordetal97].

9.8 EXERCISES

1. Consider the set $X = \{x \in \mathbb{Z}_+^2 : x_1 - x_2 \geq -1, 2x_1 + 6x_2 \leq 15, x_1 - x_2 \leq 3, 2x_1 + 4x_2 \leq 7\}$. List and represent graphically the set of feasible points. Use this to find a minimal description of $\text{conv}(X)$.

2. Let $X = \{x \in \mathbb{R}^N : \sum_{j \in N} a_j x_j \leq b\}$ with $a_j > 0$ for $j \in N$ and $b > 0$. Show that a valid inequality $\sum_{j \in N} \pi_j x_j \leq \pi_0$ with $\pi_0 > 0$ and $\pi_j < 0$ for $j \in T \subseteq N, T \neq \emptyset$ is dominated by the valid inequality $\sum_{j \in N} \max\{\pi_j, 0\} x_j \leq \pi_0$.

3. In each of the examples below a set X and a point x^* are given. Find a valid inequality for X cutting off x^* .

(i)
$$X = \{x \in B^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$$

$$x = (0, \frac{5}{8}, \frac{3}{4}, \frac{3}{4}, 0),$$

(ii)
$$X = \{x \in B^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$$

$$x = (\frac{1}{4}, \frac{1}{8}, \frac{3}{4}, \frac{3}{4}, 0),$$

(iii)
$$X = \{x \in B^5 : 7x_1 + 6x_2 + 6x_3 + 4x_4 + 3x_5 \leq 14\}$$

$$x = (\frac{1}{7}, 1, \frac{1}{2}, \frac{1}{4}, 1),$$

(iv)
$$X = \{x \in B^5 : 12x_1 - 9x_2 + 8x_3 + 6x_4 - 3x_5 \leq 2\}$$

$$x = (0, 0, \frac{1}{2}, \frac{1}{6}, 1).$$

4. Consider the knapsack set

$$X = \{x \in B^6 : 12x_1 + 9x_2 + 7x_3 + 5x_4 + 5x_5 + 3x_6 \leq 14\}.$$

Set $x_1 = x_2 = x_4 = 0$, and consider the cover inequality $x_3 + x_5 + x_6 \leq 2$ that is valid for $X' = X \cap \{x : x_1 = x_2 = x_4 = 0\}$.

(i) Lift the inequality to obtain a strong valid inequality $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_4 x_4 + x_3 + x_5 + x_6 \leq 2$ for X .

(ii*) Show that $x_3 + x_5 + x_6 \leq 2$ defines a facet of X' .

(iii*) Use (ii) and the calculations in (i) to show that the inequality obtained in (i) defines a facet of $\text{conv}(X)$.

5. Consider the set $X = \{(x, y) \in R_+^4 \times B^4 : x_1 + x_2 + x_3 + x_4 \geq 36,$
 $x_1 \leq 20y_1, x_2 \leq 10y_2, x_3 \leq 10y_3, x_4 \leq 8y_4\}$

(i) Derive a valid inequality that is a 0-1 knapsack constraint.

(ii) Use this to cut off the fractional point $x^* = (20, 10, 0, 6), y^* = (1, 1, 0, \frac{3}{4})$ with an inequality involving only y variables.

6. In each of the examples below a set X and a point (x^*, y^*) are given. Find a valid inequality for X cutting off (x^*, y^*) .

(i) $X = \{(x, y) \in R_+^3 \times B^3 : x_1 + x_2 + x_3 \leq 7, x_1 \leq 3y_1, x_2 \leq 5y_2, x_3 \leq 6y_3\}$
 $(x^*, y^*) = (2, 5, 0, \frac{2}{3}, 1, 0).$

(ii) $X = \{(x, y) \in R_+^3 \times B^3 : 7 \leq x_1 + x_2 + x_3, x_1 \leq 3y_1, x_2 \leq 5y_2, x_3 \leq 6y_3\}$
 $(x^*, y^*) = (2, 5, 0, \frac{2}{3}, 1, 0).$

(iii) $X = \{(x, y) \in R_+^6 \times B^6 : x_1 + x_2 + x_3 \leq 4 + x_4 + x_5 + x_6, x_1 \leq 3y_1, x_2 \leq 3y_2, x_3 \leq 6y_3, x_4 \leq 3y_4, x_5 \leq 5y_5, x_6 \leq 1y_6\}$.

$$(x^*, y^*) = (3, 3, 0, 0, 2, 0; 1, 1, 0, 0, \frac{2}{5}, 0).$$

7. Consider the set $X = \{(x, y_2) \in R_+^3 \times B^1 : x_1 + x_2 \leq b + x_3, x_2 \leq M y_2\}$. Derive a flow cover inequality with $C = (\{2\}, \emptyset)$. For the constraints of ULS, $s_{t-1} + x_t = d_t + s_t, x_t \leq M y_t, x_t, s_{t-1}, s_t \geq 0, y_t \in B^1$, what is the resulting inequality?

8. Consider the set $X = \{(x_1, y_2, y_3, y_4) \in R_+^1 \times B^3 : x_1 \leq 1000y_2 + 1500y_3 + 2200y_4, x_1 \leq 2000\}$.

Show that it can be rewritten as a mixed 0-1 set

$$X' = \{(x, y) \in R_+^4 \times B^4 : x_1 \leq x_2 + x_3 + x_4,$$

$$x_1 \leq 2000y_1, x_2 \leq 1000y_2, x_3 \leq 1500y_3, x_4 \leq 2200y_4\}$$

with the additional constraints

$$y_1 = 1, x_2 \geq 1000y_2, x_3 \geq 1500y_3, x_4 \geq 2200y_4.$$

Use this to find a valid inequality for X cutting off

$$(x_1, y_2, y_3, y_4) = (2000, 0, 1, \frac{5}{22}).$$

9. X is the set of incidence vectors of STSP tours. Find a valid inequality for $\text{conv}(X)$ cutting off the point x^* given in Figure 9.7.

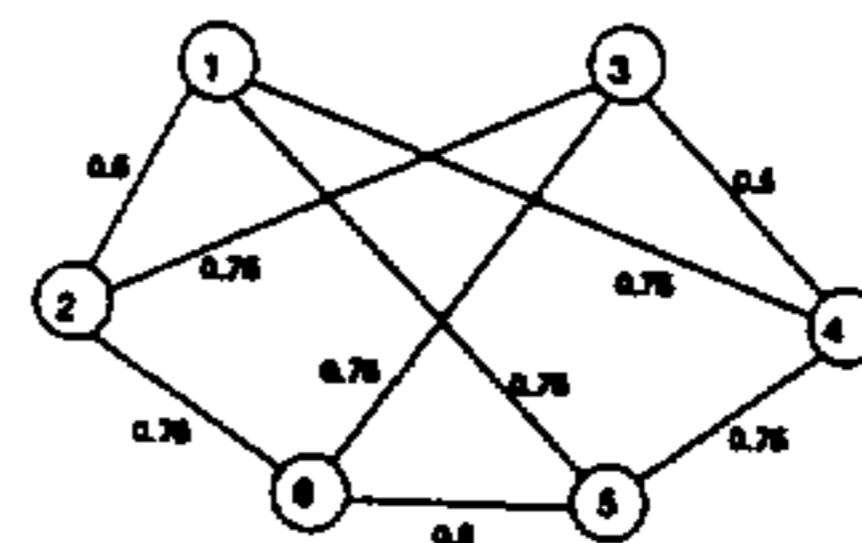


Fig. 9.7 Fractional solution of STSP

10. Solve the optimal subtour problem with $n = 6, f = (0, 10, 13, 1, 8, 12)$ and

$$(c_e) = \begin{pmatrix} - & 2 & 8 & 16 & 3 & 4 \\ & - & 6 & 4 & 3 & 8 \\ & & - & 10 & 12 & 9 \\ & & & - & 5 & 7 \\ & & & & - & 13 \\ & & & & & - \end{pmatrix}$$

11. Solve the quadratic 0-1 problem

$$\max_{x \neq 0} \{x \in B^5 : x_1x_2 + 2x_1x_3 + 3x_1x_4 + 8x_1x_5 + 1x_2x_4 + 3x_2x_5 + 6x_3x_4 + 5x_4x_5 - x_1 - 4x_2 - 10x_3 - 2x_4 - 2x_5\}.$$

12. Consider the optimal subtree of a tree problem (Section 5.3) with an additional budget constraint, with feasible region: $X = \{x \in B^n : x_j \leq x_{p(j)}$ for $j \neq 1, \sum_{j \in N} a_j x_j \leq b\}$ where $p(j)$ is the predecessor of j in the tree rooted at node 1. $C \subseteq N$ is a tree cover if the nodes of C form a subtree rooted at node 1 and $\sum_{j \in C} a_j > b$.

Show that if C is a tree cover, the inequality $\sum_{j \in C} (x_{p(j)} - x_j) \geq 1$ is valid for X where $x_{p(1)} = 1$.

13. A possible formulation of the capacitated facility location problem is:

$$\begin{aligned} \sum_{j \in N} x_{ij} &= a_i \text{ for } i \in M \\ \sum_{i \in M} x_{ij} &\leq b_j y_j \text{ for } j \in N \\ x &\in R_+^{mn}, y \in B^n. \end{aligned}$$

Given an instance with $m = 5$ and $n = 3$, use the substitution $v_j = \sum_{i \in M} x_{ij}$,

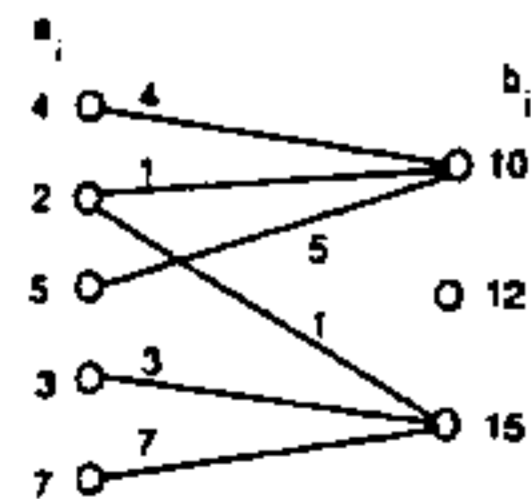


Fig. 9.8 Fractional solution of CFL

and $\sum_{j \in N} v_j = \sum_{i \in M} a_i$ to obtain an inequality cutting off the fractional solution consisting of the flow x^* shown in Figure 9.8 and $y^* = (1, \frac{4}{15}, \frac{11}{15})$.

14. Given a graph $G = (V, E)$ with $n = |V|$, consider the set of incidence vectors of the stable sets $X = \{x \in B^n : x_i + x_j \leq 1 \text{ for } e = (i, j) \in E\}$. Show that the clique inequality $\sum_{i \in C} x_i \leq 1$ is valid and defines a facet of $\text{conv}(X)$, where a clique $C \subseteq V$ is a maximal set of nodes with the property

that for all pairs $i, j \in C$, the edge $e = (i, j)$ is in E .

15.* Show that inequalities at least as strong as the cover and extended cover inequalities can be obtained by one application of the Chvátal-Gomory procedure of Section 8.3.

16. Consider an instance of the generalized assignment problem

$$\begin{aligned} \max \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \text{ for } i = 1, \dots, m \\ \sum_{i=1}^m a_{ij} x_{ij} &\leq b_j \text{ for } j = 1, \dots, n \\ x &\in B_+^{mn} \end{aligned}$$

with $m = 10, n = 5, b = (80, 63, 75, 98, 59)$ and

$$(a_{ij}) = \begin{pmatrix} 3 & 24 & 53 & 27 & 17 \\ 15 & 23 & 43 & 74 & 23 \\ 54 & 43 & 27 & 21 & 36 \\ 92 & 83 & 45 & 35 & 23 \\ 19 & 10 & 33 & 43 & 12 \\ 91 & 55 & 32 & 26 & 23 \\ 15 & 25 & 35 & 37 & 28 \\ 47 & 43 & 35 & 32 & 37 \\ 34 & 23 & 52 & 46 & 43 \\ 35 & 23 & 34 & 25 & 40 \end{pmatrix}, (c_{ij}) = \begin{pmatrix} 15 & 44 & 76 & 43 & 34 \\ 19 & 23 & 45 & 46 & 34 \\ 10 & 6 & 3 & 23 & 15 \\ 60 & 45 & 34 & 36 & 23 \\ 21 & 12 & 34 & 44 & 10 \\ 67 & 65 & 34 & 20 & 37 \\ 23 & 34 & 44 & 47 & 32 \\ 23 & 25 & 32 & 15 & 27 \\ 15 & 13 & 23 & 24 & 34 \\ 10 & 15 & 23 & 12 & 13 \end{pmatrix}$$

Solve with a mixed integer programming system. Add valid inequalities so as to reduce the number of nodes.

Describe a cutting plane algorithm for *MIP* based on this reformulation.

6. Consider the problem of scheduling n jobs on m identical machines. The processing time of job j is p_j for $j = 1, \dots, n$, and the objective is to terminate all jobs as soon as possible. Formulate as an *IP* and discuss algorithmic options for the problem.

7. Suppose that 15 pieces of length 32, 35 of length 20, 17 of length 15, and 42 of length 11 must be cut from sheets of length 104. Find the minimum number of sheets required. Formulate, discuss possible algorithms, and present lower and upper bounds on the minimum value.

12

Heuristic Algorithms

12.1 INTRODUCTION

Given that many, if not most, of the practical problems that we wish to solve are *NP*-hard, it is not surprising that heuristic or approximation algorithms play an important role in "solving" discrete optimization problems—the idea being to hopefully find a "good" feasible solution quickly.

Different reasons may lead one to choose a heuristic:

A solution is required rapidly, within a few seconds or minutes.

The instance is so large and/or complicated that it cannot be formulated as an *IP* or *MIP* of reasonable size.

Even though it has been formulated as an *MIP*, it is difficult or impossible for the branch-and-bound system to find (good) feasible solutions.

For certain combinatorial problems such as vehicle routing and machine scheduling, it is easy to find feasible solutions by inspection or knowledge of the problem structure, and a general-purpose mixed integer programming approach is ineffective.

In designing and using a heuristic, there are various questions one can ask:

Should one just accept any feasible solution, or should one ask *a posteriori* how far it is from optimal?

Can one guarantee *a priori* that the heuristic will produce a solution within ϵ (or $\alpha\%$) of optimal?

Can one say *a priori* that, for the class of problems considered, the heuristic will on average produce a solution within $\alpha\%$ of optimal?

The rest of this chapter is divided into four parts. In the first, we formalize the greedy and local exchange heuristics introduced by example in Chapter 2. We then consider two improved local exchange heuristics, tabu search and simulated annealing, that include ways to escape from a local optimum, and genetic algorithms that work with families of solutions. These heuristics, though often very effective, provide no (dual) performance bounds and thus no direct way of assessing the quality of the solutions found. We then consider some problems for which by simple analysis *a priori* worst-case bounds can be obtained. Though these bounds are typically weak, only guaranteeing solutions within, say 50%, of optimal, the resulting heuristics are typically much more effective in practice. Finally we discuss how to use a mixed integer programming system in heuristic fashion with the aim of finding good solutions quickly, and also of obtaining at least some *a posteriori* performance guarantees.

12.2 GREEDY AND LOCAL SEARCH REVISITED

Here we formalize the greedy and local search algorithms presented by example in Section 2.6. First we suppose that the problem can be written as a combinatorial problem in the form:

$$\min_{S \subseteq N} \{c(S) : v(S) \geq k\}.$$

For example, the 0-1 knapsack problem

$$\min \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \geq b, x \in B^n \right\}$$

with $c_j, a_j \geq 0$ for $j = 1, \dots, n$ is of this form with $c(S) = \sum_{j \in S} c_j$, $v(S) = \sum_{j \in S} a_j$ and $k = b$. The uncapacitated facility location problem also fits this model if we take $c(S) = \sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j$ for $S \neq \emptyset$, $v(S) = |S|$ and $k = 1$.

Below we assume that the empty set is infeasible.

A Greedy Heuristic

1. Set $S^0 = \emptyset$ (start with the empty set). Set $t = 1$.
2. Set $j_t = \arg \min \frac{c(S^{t-1} \cup \{j_t\}) - c(S^{t-1})}{v(S^{t-1} \cup \{j_t\}) - v(S^{t-1})}$ (choose the element whose additional cost per unit of resource is minimum).
3. If the previous solution S^{t-1} is feasible, and the cost has not decreased, stop with $S^G = S^{t-1}$.

4. Otherwise set $S^t = S^{t-1} \cup \{j_t\}$. If the solution is now feasible, and the cost function is nondecreasing or $t = n$, stop with $S^G = S^t$.
5. Otherwise if $t = n$, no feasible solution has been found. Stop.
6. Otherwise set $t \leftarrow t + 1$, and return to 2.

Example 12.1 We apply the greedy heuristic to an instance of the uncapacitated facility location problem with $m = 6$ clients, $n = 4$ depots, and costs

$$(c_{ij}) = \begin{pmatrix} 6 & 2 & 3 & 4 \\ 1 & 9 & 4 & 11 \\ 15 & 2 & 6 & 3 \\ 9 & 11 & 4 & 8 \\ 7 & 23 & 2 & 9 \\ 4 & 3 & 1 & 5 \end{pmatrix} \text{ and } f = (21, 16, 11, 24).$$

The algorithm gives:

Initialization. $S^0 = \emptyset$. S^0 is infeasible.

Iteration 1. $c(1) = (6 + 1 + 15 + 9 + 7 + 4) + 21 = 63$, $c(2) = 66$, $c(3) = 31$, $c(4) = 64$, so $j_1 = 3$, $S^1 = \{3\}$ and $c(S^1) = 31$. S^1 is feasible.

Iteration 2. $c(1, 3) - c(3) = (\min\{6, 3\} + \min\{1, 4\} + \min\{15, 6\} + \min\{9, 4\} + \min\{7, 2\} + \min\{4, 1\}) + (21 + 11) - 31 = 18$, $c(2, 3) - c(3) = 11$, $c(3, 4) - c(3) = 21$.

Termination. As the cost has not decreased, the algorithm stops with heuristic solution $S^G = \{3\}$ and cost 31. ■

Greedy heuristics have to be adapted to the particular problem structure. For *STSP* there are several possible greedy heuristics that choose edges one after another until a tour is obtained. The "nearest neighbor" heuristic starts from some arbitrary node, and then greedily constructs a path out from that node. The "pure greedy" heuristic chooses a least-cost edge j_t such that S^t is still part of a tour (i.e., S^t consists of a set of disjoint paths, until the last edge chosen forms a tour).

To describe local search it is simpler to formulate the combinatorial optimization problem as

$$\min_{S \subseteq N} \{c(S) : g(S) = 0\}$$

where $g(S) = 0$ represents a measure of the infeasibility of set S . Thus the constraint $v(S) \geq k$ used above can be represented here by $g(S) = (k - v(S))^+$.

For a local search algorithm, we need to define a *solution*, a *local neighborhood* $U(S)$ for each solution $S \subseteq N$, and a *goal function* $f(S)$ which can either

be just equal to $c(S)$ when S is feasible, and infinite otherwise, or a composite function of the form $c(S) + \alpha g(S)$ consisting of a weighted combination of the objective function value and a positive multiple α of the infeasibility measure for S .

A Local Search Heuristic. Choose an initial solution S . Search for a set $S' \in Q(S)$ with $f(S') < f(S)$. If none exists, stop. $S^H = S$ is a local optimum solution.

Otherwise set $S = S'$, and repeat.

Appropriate choices of neighborhood depend on the problem structure. A very simple neighborhood is that in which just one element is added or removed from S , that is, $Q(S) = \{S' : S' = S \cup \{j\} \text{ for } j \in N \setminus S\} \cup \{S' : S' = S \setminus \{i\} \text{ for } i \in S\}$. This neighborhood has only $O(n)$ elements.

Another neighborhood that is appropriate if feasible sets all have the same size is that in which one element of S is replaced by another element not in S , that is, $Q(S) = \{S' : S' = S \cup \{j\} \setminus \{i\} \text{ for } j \in N \setminus S \text{ and } i \in S\}$. This neighborhood has $O(n^2)$ elements.

For *STSP*, there is no tour differing from an initial tour by a single edge. However, if two edges are removed, there is exactly one other tour containing the remaining edges (see Figure 12.1). This leads to the well-known 2-exchange heuristic for the *STSP* on a complete graph.

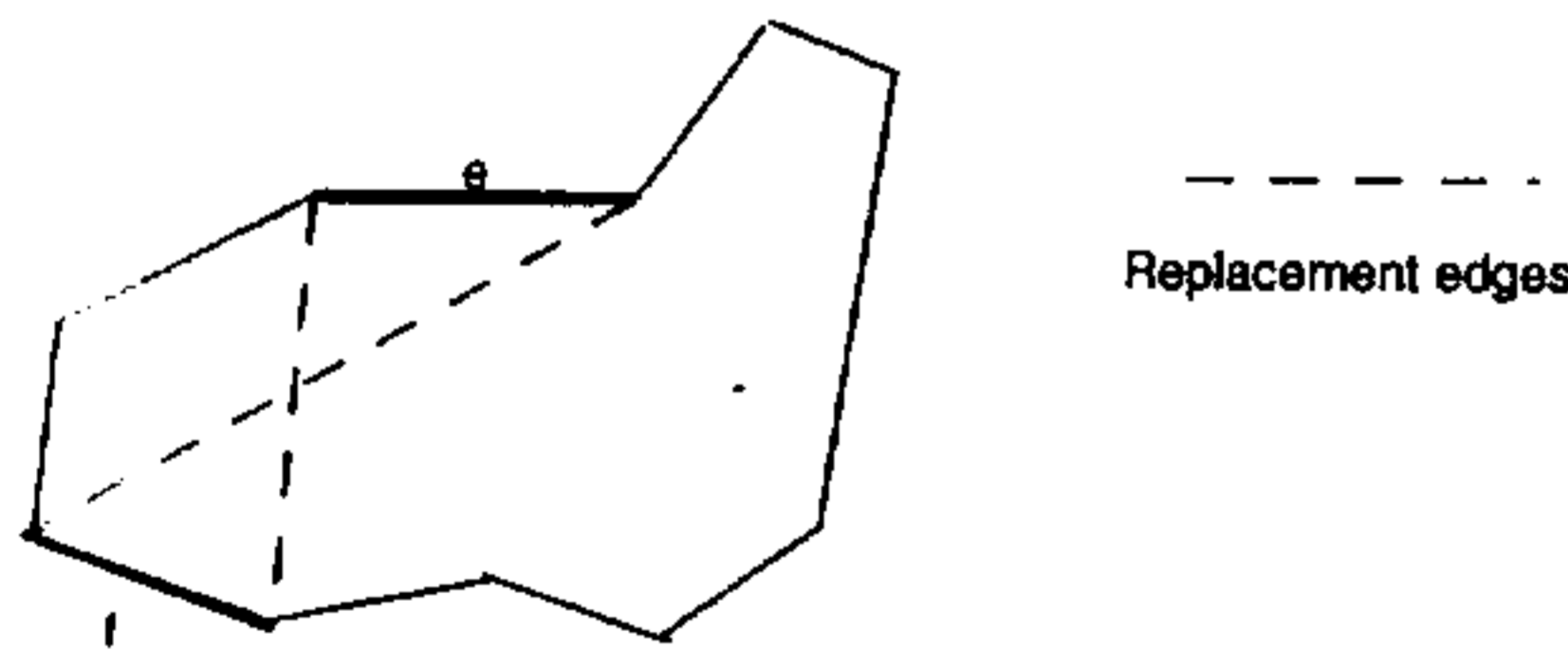


Fig. 12.1 2-Exchange for STSP

2-Exchange Heuristic for STSP

The local search heuristic is applied with the following specifications:

A set $S \subseteq E$ is a solution if the set of edges S form a tour.

$Q(S) = \{S' \text{ is a solution: } S' \neq S, |S' \cap S| = n - 2\}$, where $n = |V|$.

$f(S) = \sum_{e \in S} c_e$.

The resulting local search solution is called a *local optimum tour*.

12.3 IMPROVED LOCAL SEARCH HEURISTICS

How do we escape from a local minimum, and thus potentially do better than a local search heuristic? This is the question addressed by the tabu and simulated annealing heuristics that we now present briefly.

12.3.1 Tabu Search

When at a local minimum, a natural idea is to move to the best solution in the neighborhood even though its value is worse. One obvious difficulty with this idea is that cycling may occur, that is, the algorithm returns to the same solution every two or three steps: $S^0 \rightarrow S^1 \rightarrow S^0 \rightarrow S^1 \dots$

To avoid such cycling, certain solutions or moves are *forbidden* or *tabu*. Directly comparing the new solution with a list of all previous incumbents would require much space and be very time consuming. Instead, a *tabu list* of recent solutions, or recent solution modifications, is kept.

A basic version of the tabu search algorithm can be described as follows:

1. Initialize an empty tabu list.
2. Get an initial solution S .
3. While the stopping criterion is not satisfied:
 - 3.1. Choose a subset $Q'(S) \subseteq Q(S)$ of non-tabu solutions.
 - 3.2. Let $S' = \arg \min \{f(T) : T \in Q'(S)\}$.
 - 3.3. Replace S by S' and update the tabu list.
4. On termination, the best solution found is the heuristic solution.

The parameters specific to tabu search are:

(i) The choice of subset $Q'(S)$. Here if $Q(S)$ is small, one takes the whole neighborhood, while if $Q(S)$ is large, $Q'(S)$ can be a fixed number of neighbors of S , chosen randomly or by some heuristic rule.

(ii) The tabu list consists of a small number t of most recent solutions or modifications. If $t = 1$ or 2, it is not surprising that cycling is still common. The magic value $t = 7$ is often cited as a good choice.

(iii) The stopping rule is often just a fixed number of iterations, or a certain number of iterations without any improvement of the goal value of the best solution found.

Considering the neighborhood function

$$Q(S) = \{T \subseteq V : T = S \cup \{j\} \text{ for } j \in V \setminus S\} \cup \{T \subseteq V : T = S \setminus \{i\} \text{ for } i \in S\}$$

consisting of single element switches, the tabu list might be a list of the last t elements $\{i_1, \dots, i_t\}$ to be added to an incumbent, and of the last t elements $\{j_1, \dots, j_t\}$ to be removed. A neighbor T is then tabu if $T = S \setminus \{i_q\}$ for

some $q = 1, \dots, t$ or if $T = S \cup \{j_q\}$ for some $q = 1, \dots, t$. Therefore one cannot remove one of the t elements added most recently, and one cannot add one of the t elements removed most recently.

Tabu search also uses common sense. There is no justification to make a solution tabu if it is the best solution found to date, or it is interesting for some reason. So one or more *aspiration levels* can be defined that are used to overrule the tabu criteria. More generally, tabu search can be viewed as a search strategy that tries to take advantage of the history of the search and the problem structure intelligently.

12.3.2 Simulated Annealing

Simulated annealing is less direct. The basic idea is to choose a neighbor randomly. The neighbor then replaces the incumbent with probability 1 if it has a better goal value, and with some probability strictly between 0 and 1 if it has a worse goal value.

The probability of accepting a worse solution is proportional to the difference in goal values, so slightly worse solutions have a high probability of being accepted, while much worse solutions will only be accepted infrequently. Therefore if the number of iterations is sufficiently large, it means that one can move away from any local minimum. On the other hand, for the process to converge in the long run, the probability of accepting worse solutions decreases over time, so the algorithm should end up converging to a "good" local minimum.

A Simulated Annealing Heuristic

1. Get an initial solution S .
2. Get an initial temperature T and a reduction factor r with $0 < r < 1$.
3. While not yet frozen, do the following:
 - 3.1 Perform the following loop L times:
 - 3.1.1 Pick a random neighbor S' of S .
 - 3.1.2 Let $\Delta = f(S') - f(S)$.
 - 3.1.3 If $\Delta \leq 0$, set $S = S'$.
 - 3.1.4 If $\Delta > 0$, set $S = S'$ with probability $e^{-\Delta/T}$.
 - 3.2 Set $T \leftarrow rT$. (Reduce the temperature.)
4. Return the best solution found.

Note that as specified above, the larger Δ is, the less chance there is of making a move to a solution worse by Δ . Also as the temperature decreases, the chances of making a move to a worse solution decrease.

Exactly as for local exchange heuristics, one has to define:

- (i) A solution

- (ii) The neighbors of a solution
- (iii) The cost of a solution
- (iv) How to determine an initial solution.

The other parameters specific to simulated annealing are then:

- (v) The initial temperature T
- (vi) The cooling ratio r
- (vii) The loop length L
- (viii) The definition of "frozen," or the stopping criterion.

As application, we again consider the graph equipartition problem (see Section 2.6). There we defined a solution S to be an equipartition $(S, V \setminus S)$ with the two sets differing in size by at most one, and a neighborhood $Q(S) = \{T \subset V : |T \setminus S| = |S \setminus T| = 1\}$.

Here we go for more flexibility, by allowing any set $S \subseteq V$ representing the partition $(S, V \setminus S)$ to be a solution.

The neighborhood of a solution S is defined by a single element switch with $Q(S)$ as in Section 12.3.1. The cost of a partition is

$$f(S) = \sum_{e \in \delta(S, V \setminus S)} c_e + \alpha(|S| - |V \setminus S|)^2$$

for some $\alpha > 0$. Therefore any disparity in the size of the two sets is penalized in the goal function.

In designing and discussing improved local search algorithms, three more general concepts are useful in thinking about the right combination of choices.

Communication. It is important that the neighborhood structure be such that it is possible to get from any solution S to any other solution S' preferably in a small number of moves. Failing this, it should be possible to get from any solution S to at least one optimal solution.

Diversification. This relates to facilitating movement between very different areas of the search space. A high initial temperature T , a long tabu list, and the possibility of using random restarts all encourage diversification.

Intensification. This relates to the opposite idea of increasing the search effort in promising areas of the search space. Choosing optimally in the neighborhood, or enlarging the set $Q'(S)$ of neighbors temporarily, are measures of intensification.

12.3.3 Genetic Algorithms

Rather than working to improve individual solutions, genetic algorithms work with a finite population (set of solutions) S_1, \dots, S_k , and the population evolves (changes somewhat randomly) from one generation (iteration) to the next.

An iteration consists of the following steps:

- (i) **Evaluation.** The fitness of the individuals is evaluated.
- (ii) **Parent Selection.** Certain pairs of solutions (parents) are selected based on their fitness.
- (iii) **Crossover.** Each pair of parents combines to produce one or two new solutions (offspring).
- (iv) **Mutation.** Some of the offspring are randomly modified.
- (v) **Population Selection.** Based on their fitness, a new population is selected replacing some or all of the original population by an identical number of offspring.

We now indicate briefly ways in which the different steps can be carried out.

Evaluation. As in the local search algorithms, a goal function $f(S)$ or a pair of objective and infeasibility functions $c(S)$ and $g(S)$ are used to measure the fitness of a solution S .

Parent Selection. The idea is to choose "fitter" solutions with a higher probability. Thus from the initial population, S_i is chosen with probability $\frac{f(S_i)}{\sum_{j=1}^k f(S_j)}$.

The implementation of crossover and mutation are more problem dependent.

Crossover. Here one seeks some natural way to combine two fit solutions to produce a new fit solution. One way this is often done is by representing the solution S as a binary or integer string $x_1x_2 \dots x_r$.

Three possible ways to combine such strings are:

1-point crossover. Given two strings $x_1x_2 \dots x_r$ and $y_1y_2 \dots y_r$, and an integer $p \in \{1, \dots, r-1\}$, the two children are $x_1 \dots x_{p-1}y_p \dots y_r$ and $y_1 \dots y_{p-1}x_p \dots x_r$.

2-point crossover. Given two strings $x_1x_2 \dots x_r$ and $y_1y_2 \dots y_r$, and integers $p, q \in \{1, \dots, r-1\}$ with $p < q$, the two children are $x_1 \dots x_{p-1}y_p \dots y_{q-1}x_q \dots x_r$ and $y_1 \dots y_{p-1}x_p \dots x_{q-1}y_q \dots y_r$.

Uniform crossover. Given two strings $x_1x_2 \dots x_r$ and $y_1y_2 \dots y_r$, the result is a child $z_1 \dots z_r$ where each z_i is randomly chosen from $\{x_i, y_i\}$ for $i = 1, \dots, r$.

Mutation. A simple 1-point mutation of a child $z_1 \dots z_r$ is a random choice of $p \in \{1, \dots, r\}$ and a random integer \bar{z}_p from the appropriate range giving a modified solution $z_1 \dots z_{p-1}\bar{z}_pz_{p+1} \dots z_r$. A 2-point mutation is a swap of the values z_p and z_q for some $p < q$.

Consider again the generalized assignment problem in the form:

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij}x_{ij} \tag{12.1}$$

$$\sum_{j \in N} x_{ij} = 1 \text{ for } i \in M \tag{12.2}$$

$$\sum_{i \in M} a_{ij}x_{ij} \leq b_j \text{ for } j \in N \tag{12.3}$$

$$x \in B^{|M| \times |N|} \tag{12.4}$$

S is a "solution" if its incidence vector satisfies (12.2) and (12.4). It is represented by an integer m -vector (j_1, \dots, j_m) where j_i is the job to which person i is assigned (i.e. $x_{i,j_i} = 1$ for $i \in M$).

Two fitness values are the objective value $c(S) = \sum_{i \in M} c_{i,j_i}$ and the infeasibility value $g(S) = \sum_{j \in N} (\sum_{i \in M} a_{ij}x_{ij} - b_j)^+$.

The two-point crossover and the two-point mutation described above, both lead to new solutions satisfying (12.2) and (12.4).

Finally, the objective value $c(S)$ might be used in the selection of parents, while the feasibility measure $g(S)$ may be appropriate for the selection of the new population.

These simple ideas are obviously far from covering all possibilities. For problems in which the solution is a permutation, such as *STSP* or machine scheduling, some more suitable form of crossover is necessary.

12.4 WORST-CASE ANALYSIS OF HEURISTICS

To start with, we consider a very simple example, the integer knapsack problem:

$$(IKP) \quad z = \max\{\sum_{j=1}^n c_jx_j : \sum_{j=1}^n a_jx_j \leq b, x \in Z_+^n\}$$

where $\{a_j\}_{j=1}^n, b \in Z_+$. Without loss of generality we assume that $a_j \leq b$ for $j \in N$ and $\frac{c_j}{a_j} \geq \frac{c_i}{a_i}$ for $j \in N \setminus \{1\}$.

The greedy heuristic solution for *IKP* is $x^H = (\lfloor \frac{b}{a_1} \rfloor, 0, \dots, 0)$ with value $z^H = ca^H$.

Theorem 12.1 $\frac{z^H}{z} \geq \frac{1}{2}$.

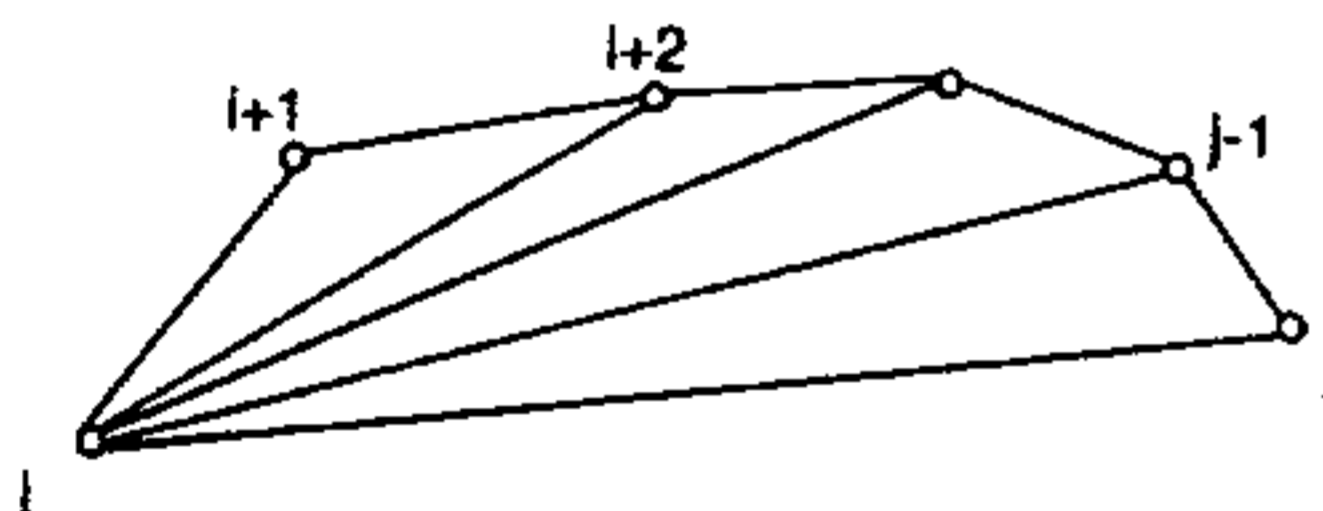
Proof. The solution to the linear programming relaxation of *IKP* is $x_1 = \frac{b}{a_1}, x_j = 0$ for $j = 2, \dots, n$ giving an upper bound $z^{LP} = \frac{c_1 b}{a_1} \geq z$.

Now as $a_1 \leq b, \lfloor \frac{b}{a_1} \rfloor \geq 1$. Setting $\frac{b}{a_1} = \lfloor \frac{b}{a_1} \rfloor + f$ with $0 \leq f < 1$, we have that $\frac{\lfloor \frac{b}{a_1} \rfloor}{\frac{b}{a_1}} = \frac{\lfloor \frac{b}{a_1} \rfloor}{\lfloor \frac{b}{a_1} \rfloor + f} \geq \frac{\lfloor \frac{b}{a_1} \rfloor}{\lfloor \frac{b}{a_1} \rfloor + \lfloor \frac{b}{a_1} \rfloor} = \frac{1}{2}$.

$$\text{So } \frac{z^H}{z} \geq \frac{z^H}{z^{LP}} = \frac{c_1 \lfloor \frac{b}{a_1} \rfloor}{c_1 \frac{b}{a_1}} = \frac{\lfloor \frac{b}{a_1} \rfloor}{\frac{b}{a_1}} \geq \frac{1}{2}.$$

It is important to observe that the analysis depends on finding both a *lower bound* on z from the heuristic solution, and also an *upper bound* on z coming from a relaxation or dual solution.

As a second problem we consider *STSP* on a complete graph with the edge lengths satisfying the *triangle inequality*, that is, if $e_i \in E$ for $i = 1, 2, 3$ are the three sides of a triangle, then $c_{e_i} + c_{e_j} \geq c_{e_k}$ for $i \neq j \neq k, i, j, k \in \{1, 2, 3\}$. Note that when this inequality holds, the shortest path between two nodes i, j is along the edge (i, j) (see Figure 12.2).



$$c_{i,j} \leq c_{i,i+1} + \dots + c_{j-1,j}$$

Fig. 12.2 Triangle inequality

To understand the heuristics and their analysis we need to introduce Eulerian graphs.

Definition 12.1 $G = (V, E)$ is a *Eulerian graph* if the degree of each node is even.

Proposition 12.2 If $G = (V, E)$ is a connected Eulerian graph and $v \in V$ is an arbitrary node, it is possible to construct a walk starting and ending at v in which each edge is traversed exactly once.

Note that a *walk* is an alternating set of nodes and edges $v_0, e_1, v_1, e_2, \dots, e_r, v_r$ where $e_i = (v_{i-1}, v_i) \in E$ for $i = 1, \dots, r$.

For the analysis below, we need the following.

Proposition 12.3 Given a complete graph H on node set V with edge lengths satisfying the triangle inequality, let $G = (V, E)$ be a connected Eulerian sub-

graph of H . Then the original graph contains a Hamiltonian (*STSP*) tour of length at most $\sum_{e \in E} c_e$.

Proof. The proof is by construction. Suppose $m = |E|$ and $v = v_0, e_1, v_1, e_2, \dots, e_m, v_m = v$ is a walk through the edges of G where each edge e_1, \dots, e_m occurs once and $e_i = (v_{i-1}, v_i)$ for $i = 1, \dots, m$. Consider the list of nodes encountered in order v_0, v_1, \dots, v_m . Suppose v_{i_k} is the k^{th} distinct node to appear in the sequence. Then $v = v_{i_1}, v_{i_2}, \dots, v_{i_n}, v_{i_{n+1}} = v$ is a tour. We now estimate its length. By the triangle property, the length of the subwalk between nodes v_{i_k} and $v_{i_{k+1}}$ is at least as long as the length of edge $f_k = (v_{i_k}, v_{i_{k+1}})$. More precisely $\sum_{j=i_k+1}^{i_{k+1}} c_{e_j} \geq c_{f_k}$. So the tour length $\sum_{k=1}^n c_{f_k} \leq \sum_{k=1}^n \sum_{j=i_k+1}^{i_{k+1}} c_{e_j} = \sum_{e \in E} c_e$.

Now we can describe a first heuristic.

The Tree Heuristic for STSP

1. In the complete graph, find a minimum-length spanning tree with edges E_T and length $z_T = \sum_{e \in E_T} c_e$.
2. Double each edge of E_T to form a connected Eulerian graph.
3. Using Proposition 12.3, convert the Eulerian graph into a tour of length z^H .

Note that in Step 1 the degree of each node of the tree is nonzero. So when the edges are duplicated in Step 2, the degree of each node is even and positive.

Proposition 12.4 $\frac{z^H}{z} \leq 2$.

Proof. As every tour consists of a tree plus an additional edge, we obtain the lower bound $z_T \leq z$. By construction, the length of the Eulerian subgraph is $2z_T$. By the tour construction procedure, $z^H \leq 2z_T$ provides the upper bound. Thus we have $\frac{z^H}{2} \leq z_T \leq z \leq z^H$.

Now we describe a second heuristic based on the construction of a shorter Eulerian subgraph. A matching is *perfect* if it is incident to every node of the graph.

The Tree/Matching Heuristic for STSP

1. In the complete graph, find a minimum-length spanning tree with edges E_T and length $z_T = \sum_{e \in E_T} c_e$.
2. Let V' be the set of nodes of odd degree in (V, E_T) . Find a perfect matching M of minimum length z_M in the graph $G' = (V', E')$ induced on V' where E' is the set of edges of E_T with both endpoints in V' . $(V, E_T \cup M)$ is a connected Eulerian graph.
3. Using Proposition 12.3, convert the Eulerian graph into a tour of length z^H .

Note that here the perfect matching M has degree 1 for each node of V' , and thus $(V, E_T \cup M)$ has positive even degree at every node.

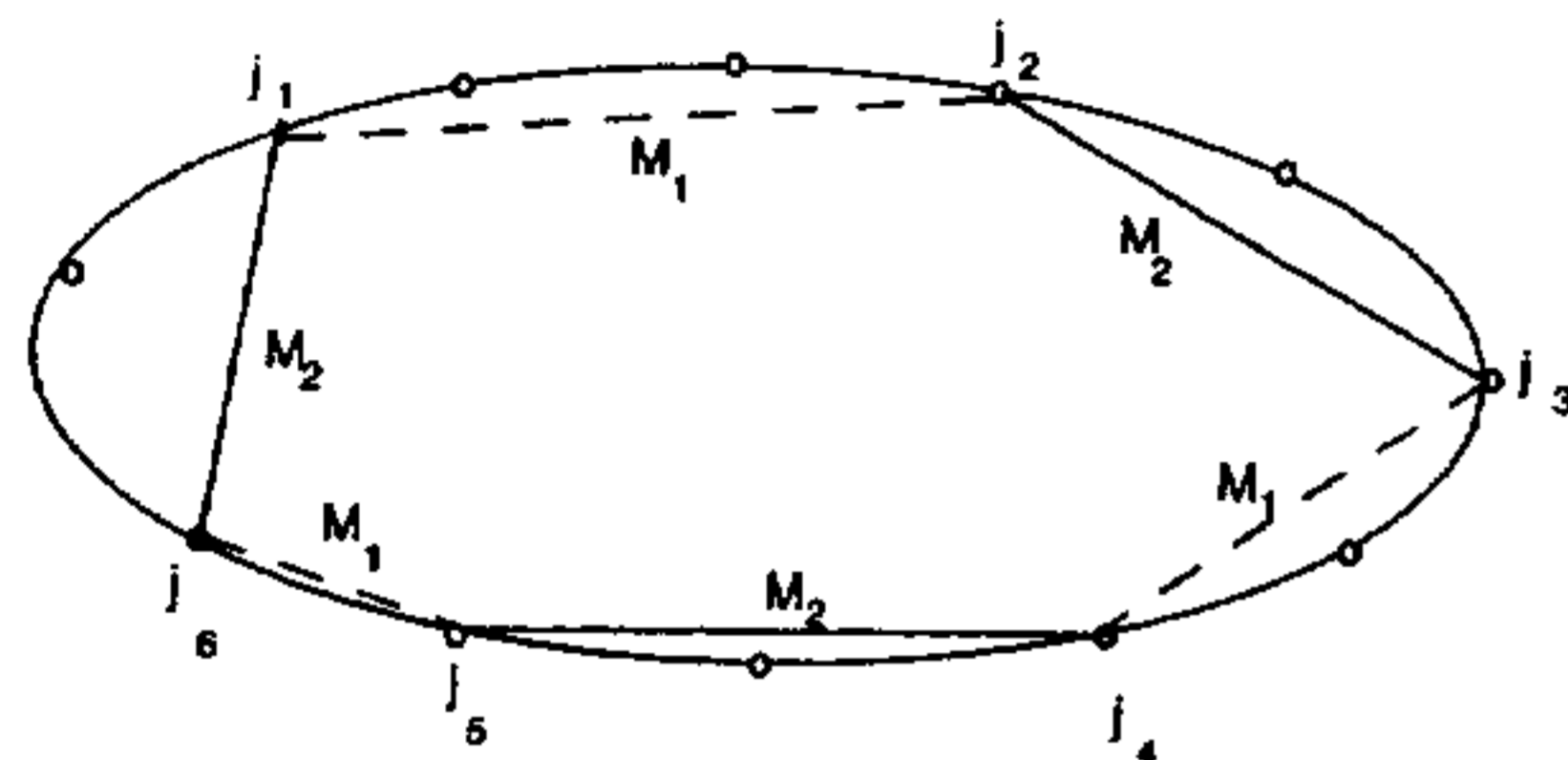


Fig. 12.3 Optimal tour longer than two matchings on V'

Proposition 12.5 $\frac{z^C}{z} \leq \frac{3}{2}$.

Proof. As above, $z_T \leq z$. Now suppose without loss of generality that the optimal tour of length z is the tour $1, 2, \dots, n$. Let j_1, j_2, \dots, j_{2k} be the nodes of V' in increasing order. Consider the perfect matchings $M_1 = \{(j_1, j_2), (j_3, j_4), \dots, (j_{2k-1}, j_{2k})\}$ of length z_{M_1} and $M_2 = \{(j_2, j_3), (j_4, j_5), \dots, (j_{2k-2}, j_{2k-1}), (j_{2k}, j_1)\}$ of length z_{M_2} , both with endpoints V' . Again by the triangle inequality $c_{j_1, j_2} + c_{j_2, j_3} + \dots + c_{j_{2k}, j_1} \leq \sum_{i=1}^{n-1} c_{i, i+1} + c_{n, 1} = z$ (see Figure 12.3). But now $z_M \leq z_{M_i}$ for $i = 1, 2$ and so $2z_M \leq z_{M_1} + z_{M_2} = c_{j_1, j_2} + c_{j_2, j_3} + \dots + c_{j_{2k}, j_1} \leq z$. Finally, again using Proposition 12.3, $z^C \leq z_T + z_M \leq z + z/2 = (3/2)z$. ■

12.5 MIP-BASED HEURISTICS

Below we discuss three heuristics for integer or mixed integer programs. The idea of the first is to take the linear programming solution at any node of the branch-and-bound tree and dive down the tree in the hope of finding a feasible solution. The other two can be implemented by anyone having access to a general mixed integer programming system.

A Dive-and-Fix Heuristic. We suppose that we have a mixed 0-1 problem. Given an linear programming solution (x^*, y^*) , let $F = \{j : y_j^* \notin \{0, 1\}\}$ be the set of 0-1 variables that are fractional.

Initialization. Take the linear programming solution (x^*, y^*) at some node.

Basic Iteration:

As long as $F \neq \emptyset$,

Let $i = \arg \min_{j \in F} \{\min\{y_j^*, 1 - y_j^*\}\}$ (find the variable closest to integer).

If $y_i^* < 0.5$, fix $y_i = 0$ (if close to 0, fix to 0).

Otherwise set $y_i = 1$ (if close to 1, fix to 1).

Solve the resulting LP.

If the LP is infeasible, stop (the heuristic has failed).

Otherwise let (x^*, y^*) be the new linear programming solution.

Termination:

If $F = \emptyset$, (x^*, y^*) is a feasible mixed integer solution.

The next two heuristics use the mixed integer system in iterative mode, because the original problem is too large or difficult. For simplicity we describe the heuristics for an IP, but the approach applies immediately to MIPs.

Suppose the problem can be written in the form:

$$(IP) \quad \begin{aligned} z &= \max c^1 x^1 + c^2 x^2 \\ A^1 x^1 + A^2 x^2 &= b \\ x^1 &\in Z_+^{n_1}, x^2 \in Z_+^{n_2} \end{aligned}$$

We suppose that the variables x_j^1 for $j \in N_1$ are more important than the variables x_j^2 for $j \in N_2$ with $n_i = |N_i|$ for $i = 1, 2$. It may be that x^1 represent major investments, while x^2 represent less important decisions such as maintenance, or else the x^1 variables may represent decisions in the initial periods, while the x^2 variables represent decisions in later periods.

The idea is to solve two (or more) easier LPs or MIPs. The first one allows us to fix or limit the range of the more important x^1 variables, whereas the second allows us to choose good values for the variables x^2 .

Relax-and-Fix Heuristic

1. *Relax.* Solve the relaxation

$$(MIP1) \quad \begin{aligned} \bar{z} &= \max c^1 x^1 + c^2 x^2 \\ A^1 x^1 + A^2 x^2 &= b \\ x^1 &\in Z_+^{n_1}, x^2 \in R_+^{n_2} \end{aligned}$$

in which the integrality of the x^2 variables is dropped. Let (\bar{x}^1, \bar{x}^2) be the corresponding solution.

2. *Fix.* Fix the important variables x^1 at their values in MIP1, and solve the restriction

Let (\bar{x}^1, \bar{x}^2) be the corresponding solution if *IP2* is feasible.

3. *Heuristic Solution.* The heuristic solution is $x^H = (\bar{x}^1, \bar{x}^2)$ with $z = cx^H \leq \bar{z}$.

The idea of the next heuristic is similar, but here we suppose that an effective strong cutting plane algorithm is available, so that after adding cuts, at least some of the integer variables take values close to integer or to their final values. Again we suppose that the variables x^1 are in some way different from, or more important than, the x^2 variables.

Cut-and-Fix Heuristic

1. *Cut.* Apply a strong cutting plane algorithm to *IP* terminating with a tightened linear programming relaxation

$$(LP1) \quad \begin{aligned} \bar{z} &= \max c^1 x^1 + c^2 x^2 \\ A^1 x^1 + A^2 x^2 &= b \\ \bar{A}^1 x^1 + \bar{A}^2 x^2 &\leq \bar{b} \\ x^1 &\in R_+^{n_1}, x^2 \in R_+^{n_2} \end{aligned}$$

with solution (\bar{x}^1, \bar{x}^2) . Here $\bar{A}^1 x^1 + \bar{A}^2 x^2 \leq \bar{b}$ represent the cuts added to the initial linear programming relaxation.

2. *Fix (or Bound).* Choose ϵ . For $j \in N_1$, set $l_j = \lfloor \bar{x}_j^1 + \epsilon \rfloor$ and $u_j = \lceil \bar{x}_j^1 - \epsilon \rceil$. Solve the restriction

$$(IP2) \quad \begin{aligned} z &= \max c^1 x^1 + c^2 x^2 \\ A^1 x^1 + A^2 x^2 &= b \\ \bar{A}^1 x^1 + \bar{A}^2 x^2 &\leq \bar{b} \\ l &\leq x^1 \leq u \\ x^1 &\in Z_+^{n_1}, x^2 \in Z_+^{n_2} \end{aligned}$$

with solution (\hat{x}^1, \hat{x}^2) if *IP2* is feasible.

3. *Heuristic Solution.* The heuristic solution is $x^H = (\hat{x}^1, \hat{x}^2)$ with $z = cx^H \leq \bar{z}$.

Observe that if ϵ is small and positive, x^1 variables taking linear programming values within ϵ of integer in *LP1* are fixed in *IP2*, while others are forced to take either the value $\lfloor \bar{x}_j^1 \rfloor$ or the value $\lceil \bar{x}_j^1 \rceil$. On the other hand, if ϵ is negative, all the x^1 variables can still take at least two values in *IP2*.

forced to take either the value $\lfloor \bar{x}_j^1 \rfloor$ or the value $\lceil \bar{x}_j^1 \rceil$. On the other hand, if ϵ is negative, all the x^1 variables can still take at least two values in *IP2*.

12.6 NOTES

For a simple treatment of the topics in Sections 12.2 and 12.3, see [Ree93]. A highly readable survey is [Pir96]. See also the annotated bibliography on local search [AarVer97]. At a more advanced level the recent book [AarLen97] is devoted to local search and its extensions, including chapters written by specialists on the complexity of local search, simulated annealing, tabu search, genetic algorithms, applications to vehicle routing, *TSP*, machine scheduling VLSI layout, and so forth. An issue of *Management Science* [FisRin88] is dedicated to the subject of heuristics.

12.2 Greedy heuristics are part of the folklore; see for instance [KueHam63] for an early application to location problems. Local search heuristics date at least to [Cro58], [ReiShe65].

12.3 Tabu search and simulated annealing have been applied successfully to find good quality feasible solutions to a remarkably wide range of problems. Tabu search started with the work of [Glo86], [Glo89], [Glo90]. The origins of simulated annealing heuristics are attributed to [Metetal53], [Kiretal83], [Cern85]. Theoretical results, relying on the asymptotic behavior of Markov chains, can be used to show that the simulated annealing algorithm almost surely terminates with a global optimum if run with a slow cooling schedule for a long enough time. However, these results are inapplicable in practice, and fail to explain the many successes of this method. A recent bibliography of simulated annealing and tabu search is [OsmLap96]. Genetic algorithms originated with the work of [Hol75], [Gol89]. The generalized assignment problem is treated in [Beas97].

Another recent approach is that of neural networks (see the texts cited above). Constraint logic programming provides an alternative approach for certain discrete problems.

12.4 The first worst-case analysis of a heuristic for a scheduling problem is in [Gra66], but the analysis of bin packing heuristics [Johetal74] really seems to have initiated much of the work in this area. The tree/matching heuristic for *STSP* is from [Chr76]; see also [CorNem78].

For certain problems it has been shown that finding a heuristic giving a performance guarantee of α or better is only possible if $\mathcal{P} = \mathcal{NP}$, while for others one can find so-called *fully polynomial approximation schemes* with performance guarantees of α for any α in a time polynomial in the input length and $1/\alpha$. The book [Hoc95] includes surveys on approximation results for bin packing, covering and packing, and network design problems among others.

11

Column Generation Algorithms

11.1 INTRODUCTION

One of the recurring ideas in optimization and in this text is that of decomposition. We have several times considered what happens when the integer programming problem (IP) $\max\{cx : x \in X\}$ has a feasible region X that can be written as the intersection of two or more sets with structure $X = \bigcap_{k=0}^K X^k$ for some $K \geq 1$. Even more particular is the case where the constraints take the form:

$$\begin{array}{rcll}
 A^1 x^1 & + A^2 x^2 & + \dots & + A^K x^K & = & b \\
 D^1 x^1 & & & & \leq & d_1 \\
 & \dots & & & \leq & \cdot \\
 & & \dots & & \leq & \cdot \\
 & & & D^K x^K & \leq & d_K \\
 x^1 \in Z_+^{n_1}, & \dots & \dots & x^K \in Z_+^{n_K} & &
 \end{array}$$

so that the sets $X^k = \{x^k \in Z_+^{n_k} : D^k x^k \leq d_k\}$ are independent for $k = 1, \dots, K$, and only the joint constraints $\sum_{k=1}^K A^k x^k = b$ link together the different sets of variables.

Given an objective function $\max \sum_{k=1}^K c^k x^k$, two earlier approaches that would permit us to benefit from such structure are cut generation, in which we would try to generate valid inequalities for each subset $X^k, k = 1, \dots, K$, and Lagrangian relaxation, in which we would dualize the joint constraints so as to obtain a dual problem:

$$\min_u L(u),$$

where

$$L(u) = \max\left\{\sum_{k=1}^K (c^k - uA^k)x^k + ub : x^k \in X^k \text{ for } k = 1, \dots, K\right\},$$

and the calculation of $L(u)$ breaks up into K distinct subproblems:

$$L(u) = \sum_{k=1}^K \max\{(c^k - uA^k)x^k : x^k \in X^k\} + ub.$$

In this chapter we examine a third way to exploit the structure of integer programs of the above form. Throughout we assume that each of the sets X^k is bounded for $k = 1, \dots, K$. The approach essentially involves solving an equivalent problem of the form:

$$\max\left\{\sum_{k=1}^K \gamma^k \lambda^k : \sum_{k=1}^K B^k \lambda^k = \beta, \lambda^k \geq 0 \text{ integer for } k = 1, \dots, K\right\}$$

where each matrix B^k has a very large number of columns, one for each of the feasible points in X^k , and each vector λ^k contains the corresponding variables.

For example, we now derive an alternative formulation of this type for the uncapacitated facility location problem *UFL*. Here the locations $j = 1, \dots, n$ correspond to the indices $k = 1, \dots, K$. For each nonempty subset $S \subseteq M$ of clients, let $\lambda_S^j = 1$ if depot j satisfies the demand of client set S . This then leads to the formulation:

$$\min \sum_{j \in N} \sum_{S \neq \emptyset} (\sum_{i \in S} c_{ij} + f_j) \lambda_S^j \tag{11.1}$$

$$\sum_{j \in N} \sum_{S \neq \emptyset, i \in S} \lambda_S^j = 1 \text{ for } i \in M \tag{11.2}$$

$$\sum_{S \neq \emptyset} \lambda_S^j \leq 1 \text{ for } j \in N \tag{11.3}$$

$$\lambda_S^j \in \{0, 1\} \text{ for } \emptyset \neq S \subseteq M, j \in N. \tag{11.4}$$

Here the cost of λ_S^j is the cost of opening depot j and serving the clients in S from depot j . The first constraints again impose that each client is served, while the second set of constraints ensure that at most one subset of clients is assigned to depot j . In practice the latter constraints are typically unnecessary. Why?

Thus the problems we wish to solve here are integer programs with an enormous number of variables, where the columns are often described implicitly as the incidence vectors of certain subsets of a set, that is tours, client subsets, and so on. Below we show how such (large) formulations of an integer program, called *Master Problems*, also arise by reformulation. We then consider how to solve the linear programming relaxation of these Master Problems, and relate the strength of this relaxation to that obtained by Lagrangian duality, or by the use of cutting planes. Finally we consider what to do when the linear programming solution is not integral, and we must resort to enumeration, leading to *IP Column Generation* or *Branch and Price* algorithms.

11.2 DANTZIG-WOLFE REFORMULATION OF AN IP

Consider the problem in the form:

$$(IP) \quad z = \max\left\{\sum_{k=1}^K c^k x^k : \sum_{k=1}^K A^k x^k = b, x^k \in X^k \text{ for } k = 1, \dots, K\right\} \tag{11.5}$$

where $X^k = \{x^k \in \mathbb{Z}_+^{n_k} : D^k x^k \leq d_k\}$ for $k = 1, \dots, K$. Assuming that each set X^k contains a large but finite set of points $\{x^{k,t}\}_{t=1}^{T_k}$, we have that $X^k =$

$$\{x^k \in \mathbb{R}^{n_k} : x^k = \sum_{t=1}^{T_k} \lambda_{k,t} x^{k,t}, \sum_{t=1}^{T_k} \lambda_{k,t} = 1, \lambda_{k,t} \in \{0, 1\} \text{ for } t = 1, \dots, T_k\}.$$

Now substituting for x^k leads to an equivalent *IP Master Problem*:

$$(IPM) \quad \begin{aligned} z &= \max \sum_{k=1}^K \sum_{t=1}^{T_k} (c^k x^{k,t}) \lambda_{k,t} \\ \sum_{k=1}^K \sum_{t=1}^{T_k} (A^k x^{k,t}) \lambda_{k,t} &= b \\ \sum_{t=1}^{T_k} \lambda_{k,t} &= 1 \text{ for } k = 1, \dots, K \\ \lambda_{k,t} &\in \{0, 1\} \text{ for } t = 1, \dots, T_k \text{ and } k = 1, \dots, K. \end{aligned}$$

Continuing with problem *UFL*, suppose we start from the weak formulation

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{j \in N} f_j y_j \tag{11.6}$$

$$\sum_{j \in N} x_{ij} = 1 \text{ for } i \in M \tag{11.7}$$

$$\sum_{i \in M} x_{ij} \leq m y_j \text{ for } j \in N \tag{11.8}$$

$$x \in B^{|M| \times |N|}, y \in B^{|N|}. \tag{11.9}$$

Here we can take (11.7) as the joint constraints, and $X^k = \{(x_{1k}, \dots, x_{mk}, y_k) : \sum_{i \in M} x_{ik} \leq m y_k, x_{ik} \in B^1 \text{ for } i \in M, y_k \in \{0, 1\}\}$. The points in X^k are $\{(x_S^k, 1)\}_{S \subseteq M}$, where x_S^k is the incidence vector of $S \subseteq M$, and $(0, 0)$ with associated variables λ_S^k, ν^k respectively leading to the *IP Master Problem*:

$$\begin{aligned} \min \sum_{j \in N} &[\sum_{S \neq \emptyset} (\sum_{i \in S} c_{ij} + f_j) \lambda_S^j + f_j \lambda_0^j] \\ \sum_{j \in N} \sum_{S \neq \emptyset, i \in S} &\lambda_S^j = 1 \text{ for } i \in M \\ \sum_{S \neq \emptyset} \lambda_S^j + \lambda_0^j + \nu^j &= 1 \text{ for } j \in N \\ \lambda_S^j &\in \{0, 1\} \text{ for } S \subseteq M, j \in N, \nu^j \in \{0, 1\} \text{ for } j \in N. \end{aligned}$$

Observe that as $f_j > 0$, variable λ_0^j is dominated by ν^j and can be dropped. Now this formulation and (11.1)–(11.4) are identical if we take ν^j to be the slack variable in (11.3).

11.3 SOLVING THE MASTER LINEAR PROGRAM

Here we use a column generation algorithm to solve the linear programming relaxation of the Integer Programming Master Problem, called the *Linear Programming Master Problem*:

$$(LPM) \quad \begin{aligned} z^{LPM} &= \max \sum_{k=1}^K \sum_{t=1}^{T_k} (c^k x^{k,t}) \lambda_{k,t} \\ \sum_{k=1}^K \sum_{t=1}^{T_k} (A^k x^{k,t}) \lambda_{k,t} &= b \\ \sum_{t=1}^{T_k} \lambda_{k,t} &= 1 \text{ for } k = 1, \dots, K \\ \lambda_{k,t} &\geq 0 \text{ for } t = 1, \dots, T_k, k = 1, \dots, K \end{aligned}$$

where there is a column $\begin{pmatrix} c^k x \\ A^k x \\ e_k \end{pmatrix}$ for each $x \in X^k$. Below we will use $\{\pi_i\}_{i=1}^m$

as the dual variables associated with the joint constraints, and $\{\mu_k\}_{k=1}^K$ as dual variables for the second set of constraints, known as *convexity constraints*.

The idea is to solve the linear program by the primal simplex algorithm. However, the pricing step of choosing a column to enter the basis must be modified because of the enormous number of columns. Rather than pricing the columns one by one, the problem of finding a column with the largest reduced price is itself a set of K optimization problems.

Initialization. We suppose that a subset of columns (at least one for each k) is available, providing a feasible *Restricted Linear Programming Master Problem*

$$(RLPM) \quad \begin{aligned} \bar{z}^{LPM} &= \max \bar{c} \bar{\lambda} \\ \bar{A} \bar{\lambda} &= \bar{b} \\ \bar{\lambda} &\geq 0 \end{aligned}$$

where $\bar{b} = \begin{pmatrix} b \\ 1 \end{pmatrix}$, \bar{A} is generated by the available set of columns and is a submatrix of

$$\begin{pmatrix} A^1 x^{1,1} & \dots & A^1 x^{1,T_1} & A^2 x^{2,1} & \dots & A^2 x^{2,T_2} & \dots & A^K x^{K,1} & \dots & A^K x^{K,T_K} \\ 1 & \dots & 1 & & & & & & & \\ & & & 1 & \dots & 1 & & & & \\ & & & & & & & & & \\ & & & & & & & & & \\ & & & & & & & 1 & \dots & 1 \end{pmatrix}$$

and $\bar{c}, \bar{\lambda}$ are the corresponding costs and variables. Solving *RLPM* gives an optimal primal solution $\bar{\lambda}^*$ and an optimal dual solution $(\pi, \mu) \in R^m \times R^K$

Primal Feasibility. Any feasible solution of *RLPM* is feasible for *LPM*. In particular, $\bar{\lambda}^*$ is a feasible solution of *LPM*, and so $\bar{z}^{LPM} = \bar{c} \bar{\lambda}^* = \sum_{i=1}^m \pi_i b_i + \sum_{k=1}^K \mu_k \leq z^{LPM}$.

Optimality Check for LPM. We need to check whether (π, μ) is dual feasible for *LPM*. This involves checking for each column, that is for each k , and for each $x \in X^k$ whether the reduced price $c^k x - \pi A^k x - \mu_k \leq 0$. Rather than examining each point separately, we treat all points in X^k implicitly by solving an optimization subproblem:

$$\zeta_k = \max \{ (c^k - \pi A^k) x - \mu_k : x \in X^k \}.$$

Stopping Criterion. If $\zeta_k = 0$ for $k = 1, \dots, K$, the solution (π, μ) is dual feasible for *LPM*, and so $z^{LPM} \leq \sum_{i=1}^m \pi_i b_i + \sum_{k=1}^K \mu_k$. As the value of the primal feasible solution $\bar{\lambda}$ equals that of this upper bound, $\bar{\lambda}$ is optimal for *LPM*.

Generating a New Column. If $\zeta_k > 0$ for some k , the column corresponding to the optimal solution \bar{x}^k of the subproblem has positive reduced price. Intro-

ducing the column $\begin{pmatrix} c^k \bar{x}^k \\ A^k \bar{x}^k \\ e_k \end{pmatrix}$ leads to a new *Restricted Linear Programming*

Master Problem that can be easily reoptimized (e.g., by the primal simplex algorithm).

A Dual (Upper) Bound. From the subproblem, we have that $\zeta_k \geq (c^k - \pi A^k) x - \mu_k$ for all $x \in X^k$. It follows that $(c^k - \pi A^k) x - \mu_k - \zeta_k \leq 0$ for all $x \in X^k$. Therefore setting $\zeta = (\zeta_1, \dots, \zeta_K)$, we have that $(\pi, \mu + \zeta)$ is dual feasible in *LPM*. Therefore

$$z^{LPM} \leq \pi b + \sum_{k=1}^K \mu_k + \sum_{k=1}^K \zeta_k.$$

These different observations lead directly to an algorithm for *LPM* that terminates when $\zeta_k = 0$ for $k = 1, \dots, K$. However, as in Lagrangian relaxation, it may be possible to terminate earlier.

An Alternative Stopping Criterion. If the subproblem solutions $(\bar{x}^1, \dots, \bar{x}^K)$ satisfy the original joint constraints $\sum_{k=1}^K A^k \bar{x}^k = b$, then $(\bar{x}^1, \dots, \bar{x}^K)$ is optimal.

This follows because $\zeta_k = (c^k - \pi A^k) \bar{x}^k - \mu_k$ implies that $\sum_k c^k \bar{x}^k = \sum_k \pi A^k \bar{x}^k + \sum_k \mu_k + \sum_k \zeta_k = \pi b + \sum_k \mu_k + \sum_k \zeta_k$. Therefore the primal feasible solution has the same value as the upper bound on z^{LPM} .

11.3.1 STSP by Column Generation

Here we consider the application of the above algorithm to solve the Master Linear Program of a problem in which there is just a single subproblem. We again consider the symmetric traveling salesman problem, which can be written as

$$\min\left\{\sum_{e \in E} c_e x_e : \sum_{e \in \delta(i)} x_e = 2 \text{ for } i \in N, x \in X^1\right\}$$

where

$$X^1 = \{x \in Z_+^m : \sum_{e \in \delta(i)} x_e = 2, \sum_{e \in E(S)} x_e \leq |S| - 1 \text{ for } \emptyset \subset S \subseteq N \setminus \{1\}, \sum_{e \in E} x_e = n\}$$

is the set of incidence vectors of 1-trees.

Writing $x_e = \sum_{t: e \in E^t} \lambda_t$, where $G^t = (N, E^t)$ is the t^{th} 1-tree for $t = 1, \dots, T_1$, the degree constraints become $\sum_{e \in \delta(i)} x_e = \sum_{e \in \delta(i)} \sum_{t: e \in E^t} \lambda_t = \sum_t d_i^t \lambda_t = 2$ where d_i^t is the degree of node i in the 1-tree G^t . Thus the corresponding Linear Programming Master is

$$(LPM) \quad \begin{aligned} \min & \sum_{t=1}^{T_1} (cx^t) \lambda_t \\ \sum_{t=1}^{T_1} d_i^t \lambda_t &= 2 \text{ for } i \in N \\ \sum_{t=1}^{T_1} \lambda_t &= 1 \\ \lambda &\in R_+^{T_1} \end{aligned}$$

with which we associate dual variables $\{u_i\}_{i=1}^n$ to the degree constraints, and dual variable μ to the convexity constraint. The corresponding single subproblem is

$$\zeta_1 = \min\left\{\sum_{e \in E} (c_e - u_i - u_j) x_e - \mu : x \in X^1\right\}$$

in the 1-tree G^t has reduced cost $cx^t - \sum_{i \in N} d_i^t u_i - \mu = cx^t - \sum_{i \in N} u_i \sum_{e \in \delta(i)} x_e^t - \mu = \sum_{e \in E} (c_e - u_i - u_j) x_e^t - \mu$, where x_e^t for $e \in E$ are the edge variables of the 1-tree G^t , and $e = (i, j)$ for $e \in E$.

Note that because we are dealing with 1-trees, $d_i^t = 2$ for all t , and so the first equation in LPM is twice the convexity constraint. As a result we can drop the convexity constraint.

Example 11.1 Consider an instance of STSP with distance matrix

$$c = \begin{pmatrix} 7 & 2 & 1 & 5 \\ & 3 & 6 & 8 \\ & & 4 & 2 \\ & & & 9 \end{pmatrix}$$

We initialize with a restricted LPM having 7 columns, corresponding to a tour of length 28 and six 1-trees chosen arbitrarily

$$\begin{aligned} \min \quad & 28\lambda_1 + 25\lambda_2 + 21\lambda_3 + 19\lambda_4 + 22\lambda_5 + 18\lambda_6 + 28\lambda_7 \\ & 2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 2\lambda_4 + 2\lambda_5 + 2\lambda_6 + 2\lambda_7 = 2 \\ & 2\lambda_1 + 2\lambda_2 + 2\lambda_3 + 1\lambda_4 + 1\lambda_5 + 2\lambda_6 + 3\lambda_7 = 2 \\ & 2\lambda_1 + 3\lambda_2 + 2\lambda_3 + 3\lambda_4 + 2\lambda_5 + 3\lambda_6 + 1\lambda_7 = 2 \\ & 2\lambda_1 + 2\lambda_2 + 3\lambda_3 + 3\lambda_4 + 3\lambda_5 + 1\lambda_6 + 1\lambda_7 = 2 \\ & 2\lambda_1 + 1\lambda_2 + 1\lambda_3 + 1\lambda_4 + 2\lambda_5 + 2\lambda_6 + 3\lambda_7 = 2 \\ & \lambda \geq 0. \end{aligned}$$

The resulting linear programming solution is $\lambda = (0, 0, \frac{1}{4}, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ with cost 22.25 and dual solution $u = (\frac{151}{8}, -1, -\frac{11}{2}, -\frac{5}{4}, 0)$. The corresponding reduced cost matrix for the subproblem is

$$\begin{pmatrix} -\frac{87}{8} & -\frac{91}{8} & -\frac{133}{8} & -\frac{111}{8} \\ & \frac{19}{2} & \frac{33}{4} & 9 \\ & & \frac{43}{4} & \frac{15}{2} \\ & & & \frac{41}{4} \end{pmatrix}$$

The optimal 1-tree is $x_{14} = x_{15} = x_{24} = x_{25} = x_{35} = 1$ with $\zeta_1 = -\frac{23}{4}$. Therefore $22.25 + \zeta_1 = 16.75 \leq z^{LPM} \leq 22.25$.

We start a new iteration by introducing this 1-tree as a new column in the restricted master with cost 22, and degrees (2,2,1,2,3). The new linear programming solution is $\lambda = (0, 0, \frac{1}{3}, 0, 0, \frac{1}{3}, 0, \frac{1}{3})$ with cost 20.333 and dual solution $u = (\frac{65}{6}, \frac{1}{3}, -\frac{5}{3}, \frac{2}{3}, 0)$.

The corresponding reduced cost matrix for the subproblem is

$$\begin{pmatrix} -\frac{25}{6} & -\frac{43}{6} & -\frac{21}{2} & -\frac{35}{6} \\ & \frac{13}{3} & 5 & \frac{23}{3} \\ & & 5 & \frac{11}{3} \\ & & & \frac{25}{3} \end{pmatrix}$$

The optimal 1-tree is $x_{13} = x_{14} = x_{23} = x_{24} = x_{35} = 1$ with $\zeta_1 = -\frac{14}{3}$. The lower bound of $20.333 - \frac{14}{3} = 15.667$ is not as good as that obtained before. Therefore we now have $16.75 \leq z^{LPM} \leq 20.333$.

Again we introduce this 1-tree as a new column in the restricted master with cost 14, and degrees (2,2,3,2,1). The new linear programming solution is $\lambda = (0, 0, 0, 0, 0, 0, \frac{1}{2}, \frac{1}{2})$ with cost 18 and dual solution $u = (13, 0, -4, 0, 0)$.

The corresponding reduced cost matrix for the subproblem is

$$\begin{pmatrix} -6 & -7 & -12 & -8 \\ & 7 & 6 & 8 \\ & & 8 & 6 \\ & & & 9 \end{pmatrix}$$

The optimal 1-tree is $x_{14} = x_{15} = x_{23} = x_{24} = x_{35} = 1$ with $\zeta = -1$. The lower bound on z^{LPM} increases to $18 - 1 = 17$. As this 1-tree is a tour, it follows from the alternative stopping criterion that it is optimal. Alternatively one can check that its real cost is 17. ■

11.3.2 Strength of the Linear Programming Master

How strong is the linear programming relaxation of the Master Problem? Is there some hope that it will solve the original problem *IP*?

Proposition 11.1

$$z^{LPM} = \max \left\{ \sum_{k=1}^K c^k x^k : \sum_{k=1}^K A^k x^k = b, x^k \in \text{conv}(X^k) \text{ for } k = 1, \dots, K \right\}.$$

Proof. *LPM* can be obtained from the original problem *IP* by substituting $x^k = \sum_{t=1}^{T_k} x^{k,t} \lambda_{k,t}$, $\sum_{t=1}^{T_k} \lambda_{k,t} = 1$, $\lambda_{k,t} \geq 0$ for $t = 1, \dots, T_k$. This is equivalent to substituting $x^k \in \text{conv}(X^k)$. ■

As discussed in the introduction to this chapter, when *IP* is decomposable, Lagrangian relaxation and cutting plane algorithms are two possible alternative approaches. Specifically let w_{LD} be the value of the Lagrangian dual when the joint constraints $\sum_{k=1}^K A^k x^k = b$ are dualized, and let z^{CUT} be the value obtained when cutting planes are added to the linear programming relaxation of *IP* using an exact separation algorithm for each of the sets $\text{conv}(X^k)$ for $k = 1, \dots, K$.

The next result, showing that all three approaches are in some sense equivalent as they lead to the same dual bounds, is based on Theorem 10.3, Proposition 11.1, and the fact that an exact separation algorithm for $\text{conv}(X^k)$ implicitly generates $\text{conv}(X^k)$.

Theorem 11.2 $z^{LPM} = w_{LD} = z^{CUT}$.

As the subproblems solved in both the column generation and Lagrangian dual approaches are optimization problems over X^k , column generation can be viewed as an algorithm for solving the Lagrangian dual in which the dual variables π are updated using linear programming by solving the Restricted Linear Programming Master. This is in comparison with the subgradient algorithm often used to solve the Lagrangian dual that is based on a much simpler updating procedure.

On the other hand, if we use the cutting plane approach, though the bound obtained is potentially the same, separation problems over $\text{conv}(X^k)$ have to be solved instead of optimization problems.

As the theoretical complexity of the optimization and separation problems for $\text{conv}(X^k)$ is the same, the choice of approach depends on the relative difficulty in solving the two problems as well as on the convergence of the column generation and cutting plane algorithms in practice.

11.4 IP COLUMN GENERATION FOR 0-1 IP

If when the column generation algorithm terminates, the optimal solution vector $\bar{\lambda} = (\bar{\lambda}^1, \dots, \bar{\lambda}^K)$ of *LPM* is not integer, then *IPM* is not yet solved. However, $z^{LPM} \geq z$, which suggests the possibility of using such upper bounds in a branch-and-bound algorithm. In this section we present an algorithm for 0-1 problems using this bound, called an *IP column generation* or *branch-and-price* algorithm.

Again we have the original problem

$$(IP) \quad z = \max \left\{ \sum_{k=1}^K c^k x^k : \sum_{k=1}^K A^k x^k = b, D^k x^k \leq d_k \text{ for } k = 1, \dots, K, x^k \in B^{n_k} \text{ for } k = 1, \dots, K \right\},$$

and its reformulation

$$(IPM) \quad \begin{aligned} z &= \max \sum_{k=1}^K \sum_{t=1}^{T_k} (c^k x^{k,t}) \lambda_{k,t} \\ \sum_{k=1}^K \sum_{t=1}^{T_k} (A^k x^{k,t}) \lambda_{k,t} &= b \\ \sum_{t=1}^{T_k} \lambda_{k,t} &= 1 \text{ for } k = 1, \dots, K \\ \lambda_{k,t} &\in \{0, 1\} \text{ for } t = 1, \dots, T_k, k = 1, \dots, K. \end{aligned}$$

whose linear programming relaxation has optimal solution $\bar{\lambda}$.

Because the points $x^{k,t} \in X^k$ are distinct 0-1 vectors, note that $\bar{x}^k = \sum_{t=1}^{T_k} \bar{\lambda}_{k,t} x^{k,t} \in \{0, 1\}^{n_k}$ if and only if $\bar{\lambda}$ is integer. Therefore if $\bar{\lambda}$ is not integer, there is some κ and j such that the corresponding 0-1 variable x_j^κ has linear programming value \bar{x}_j^κ that is fractional, and on which one can branch.

This suggests the branching scheme shown in Figure 11.1(a), in which the set S of all feasible solutions is split into $S_0 = S \cap \{x : x_j^\kappa = 0\}$ and $S_1 = S \cap \{x : x_j^\kappa = 1\}$. Note that this is exactly the same type of scheme used in the basic branch-and-bound algorithm in Chapter 7.

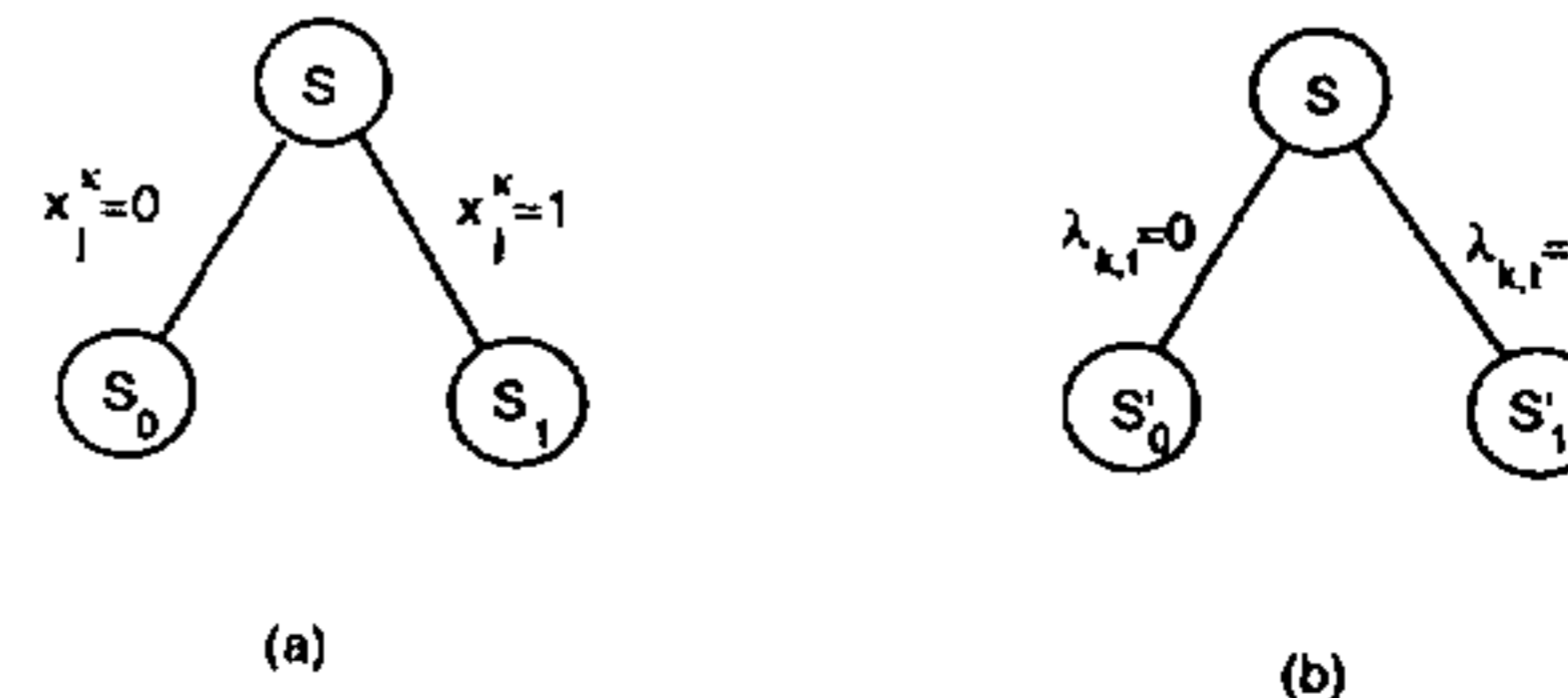


Fig. 11.1 Branching for 0-1 column generation: (a) original (b) column variables

It is important now to make sure that it will still be possible to solve the new linear programming master problems without difficulty. To do this we need

to define the new Master Problems associated with S_i for $i = 0, 1$, and the new subproblems.

Now as $x_j^k = \sum_{t=1}^{T_k} \lambda_{k,t} x_j^{k,t} \in \{0, 1\}$, $x_j^k = \delta \in \{0, 1\}$ implies that $x_j^{k,t} = \delta$ for all k, t with $\lambda_{k,t} > 0$. So the Master Problem at node $S_i = S \cap \{x : x_j^k = i\}$ for $i = 0, 1$ is

$$(IPM(S_i)) \quad z(S_i) = \max \sum_{k \neq \kappa} \sum_t (c^k x^{k,t}) \lambda_{k,t} + \sum_{t: x_j^{\kappa,t} = i} (c^\kappa x^{\kappa,t}) \lambda_{\kappa,t} \\ \sum_{k \neq \kappa} \sum_t (A^k x^{k,t}) \lambda_{k,t} + \sum_{t: x_j^{\kappa,t} = i} (A^\kappa x^{\kappa,t}) \lambda_{\kappa,t} = b \\ \sum_t \lambda_{k,t} = 1 \text{ for } k \neq \kappa \\ \sum_{t: x_j^{\kappa,t} = i} \lambda_{\kappa,t} = 1 \\ \lambda_{k,t} \in \{0, 1\} \text{ for } t = 1, \dots, T_k, k = 1, \dots, K.$$

This has the same form as the original Master Problem except that a set of columns are excluded on each branch, and the previous LPM solution is now infeasible. Turning to the column generation subproblems, the subproblem is unchanged if $k \neq \kappa$. However, for subproblem κ and $i = 0, 1$, we have

$$\zeta_\kappa(S_i) = \max\{(c^\kappa - \pi A^\kappa)x - \mu_\kappa : x \in X^\kappa, x_j = i\},$$

which is very similar to the original subproblem.

Another idea is to branch on some fractional $\lambda_{k,t}$ variable, fixing it to 0 and 1 respectively, see Figure 11.1(b). Note, however, that on the branch in which $\lambda_{k,t} = 0$, just one column, corresponding to the t^{th} solution of subproblem k , is excluded, so the resulting problem is almost identical to the original one. This means that the resulting enumeration tree has the undesirable property of being highly unbalanced. In addition it is often difficult to impose the condition $\lambda_{k,t} = 0$, and thus to prevent the same solution being generated again as optimal solution after branching.

One potential advantage of the column generation approach, visible in Example 11.2, is that the optimal solutions to RLPM are often integral or close to integral. In the first case this gives a feasible integer solution, and in the second such a solution can often be obtained by a simple rounding heuristic.

11.5 IMPLICIT PARTITIONING/PACKING PROBLEMS

An important subclass of decomposable 0-1 IPs are packing and partitioning problems. Given a finite set $M = \{1, \dots, m\}$, there are K implicitly described sets of feasible subsets, and the problem is to find a maximum value packing or partition of M consisting of certain of these subsets.

In terms of the original IP (11.5) of Section 11.2, we set $x^k = (y^k, w^k)$ with $y^k \in \{0, 1\}^m$ the incidence vector of subset k of M , $c^k = (r^k, f^k)$, $A^k = (I, 0)$ and $b = 1$. One should think of the variables w^k as auxiliary variables needed to define whether the subset with incidence vector y^k is feasible, and to define

the formulation

$$z = \max \left\{ \sum_{k=1}^K (e^k y^k + f^k w^k) : \sum_{k=1}^K y^k \leq 1, (y^k, w^k) \in X^k \text{ for } k = 1, \dots, K \right\}.$$

Now if $(y^{k,t}, w^{k,t})$ corresponds to the t^{th} feasible solution in the set X^k , and $\lambda_{k,t}$ is the corresponding variable, we obtain an equivalent Integer Programming Master

$$z = \max \sum_{k=1}^K \sum_{t=1}^{T_k} (e^k y^{k,t} + f^k w^{k,t}) \lambda_{k,t} \\ \sum_{k=1}^K \sum_{t: y_i^{k,t} \leq 1} \lambda_{k,t} \leq 1 \text{ for } i \in M \\ \sum_{t=1}^{T_k} \lambda_{k,t} = 1 \text{ for } k = 1, \dots, K \\ \lambda_{k,t} \in \{0, 1\} \text{ for } t = 1, \dots, T_k, k = 1, \dots, K.$$

We now present several problems of this type. Clearly as the partitioning problem is a special case of (11.5), the algorithm of the previous section can be applied.

Multi-Item Lot-Sizing. Suppose we are given demands d_t^k for items $k = 1, \dots, K$ over a time horizon $t = 1, \dots, T$. All items must be produced on a single machine; the machine can produce only one item in each period and has a capacity C_t^k if item k is produced in period t . Given production, storage, and set-up costs for each item in each period, we wish to find a minimum cost production plan. This problem can be formulated as

$$\min \sum_{k=1}^K \sum_{t=1}^T (p_t^k x_t^k + h_t^k s_t^k + f_t^k y_t^k) \\ \sum_{k=1}^K y_t^k \leq 1 \text{ for } t = 1, \dots, T \\ (x^k, s^k, y^k) \in X^k \text{ for } k = 1, \dots, K$$

where $X^k = \{(x^k, s^k, y^k) \in R_+^n \times R_+^n \times B^n : s_{t-1}^k + x_t^k = d_t^k + s_t^k, x_t^k \leq C_t^k y_t^k \text{ for } t = 1, \dots, T\}$.

Clustering. Given a graph $G = (V, E)$, edge costs c_e for $e \in E$, node weights d_i for $i \in V$, and a cluster capacity C , we wish to split the node set V into K (possibly empty) clusters satisfying the property that the sum of the node weights in each cluster does not exceed C , in a way that minimizes the sum of the weights of edges between clusters (maximizes the sum of weights of edges within clusters). Figure 11.2 shows a feasible solution for an instance with 3 clusters and a capacity of 9. The thick edges are those between clusters. The problem can be formulated as

$$\max \sum_{k=1}^K \sum_{e \in E} c_e w_e^k \\ \sum_{i \in V} d_i y_i^k \leq C \text{ for } k = 1, \dots, K \\ (w^k, y^k) \in X^k \text{ for } k = 1, \dots, K$$

where $X^k = \{(w^k, y^k) \in B^m \times B^n : w_e^k \leq y_i^k, w_e^k \leq y_j^k, w_e^k \geq y_i^k + y_j^k - 1 \text{ for } e = (i, j) \in E, \sum_{i \in V} d_i y_i^k \leq C\}$ with $y_i^k = 1$ if node i is in cluster k and $w_e^k = 1$ if edge e has both endpoints in cluster k .

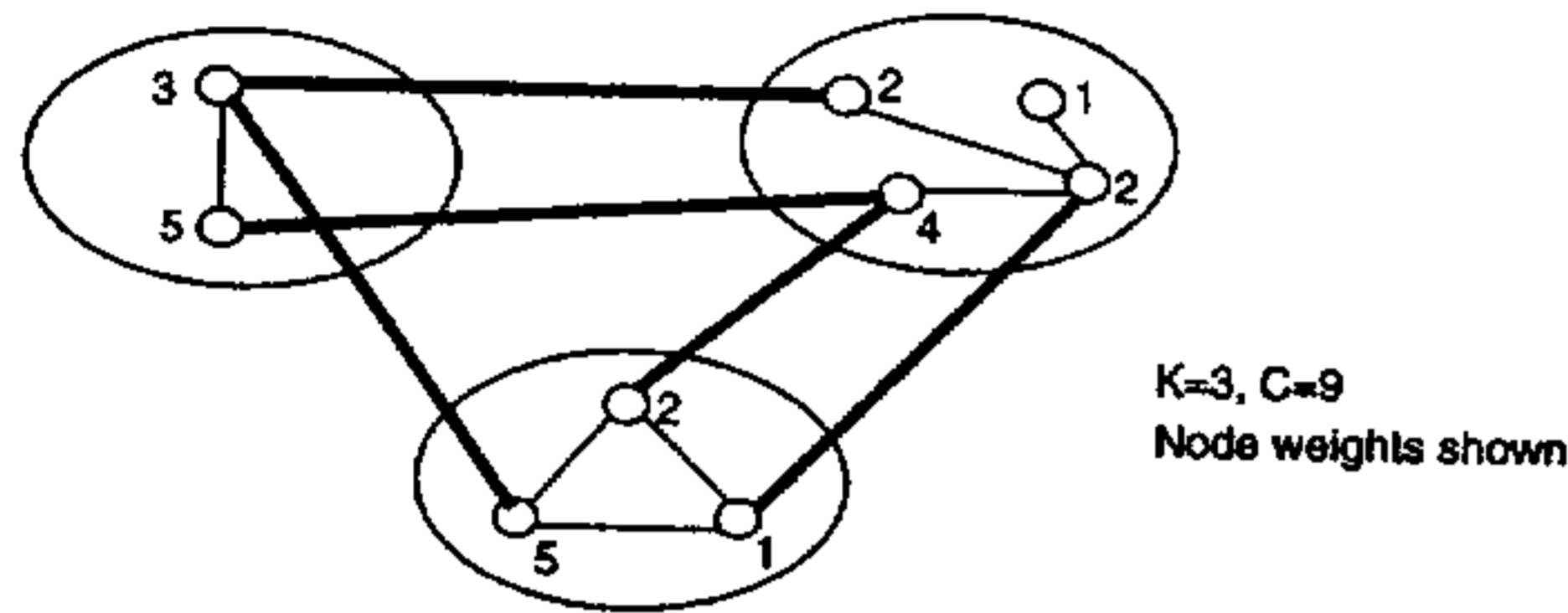


Fig. 11.2 Clustering solution with three clusters

Capacitated Vehicle Routing. Given a graph $G = (V, E)$, a depot node 0, edge costs c_e for each $e \in E$, K identical vehicles of capacity C , and client orders d_i for $i \in V \setminus \{0\}$, we wish to find a set of subtours (cycles) for each vehicle such that (i) each subtour contains the depot, (ii) together the subtours contain all the nodes, (iii) the subtours are disjoint on the node set $V \setminus \{0\}$, and (iv) the total demand on each subtour (the total amount delivered by each vehicle) does not exceed C .

Another problem with such a decomposable structure is the generalized assignment problem. An instance of *GAP* is treated by branch-and-cut in Section 9.6.

11.6 PARTITIONING WITH IDENTICAL SUBSETS*

The clustering and vehicle routing problems of the last section both have the property that the clusters or vehicles are interchangeable (independent of k). This means that the numbering of the subsets is arbitrary, and exchanging any two sets leads to an essentially identical solution.

Here we consider how the integer programming column generation algorithm of Section 11.4 can be specialized to take account of this symmetry. As $X^k = X$, $(e^k, f^k) = (e, f)$ and $T_k = T$ for all k , we can set $\lambda_t = \sum_{k=1}^K \lambda_{k,t}$ and *IPM* now takes the form:

$$\max \sum_{t=1}^T (cy^t + fw^t) \lambda_t$$

$$\sum_{t: y_i^t = 1} \lambda_t = 1 \text{ for } i \in M$$

$$\sum_{t=1}^T \lambda_t \leq K$$

$$\lambda \in B^T.$$

There is now just a single column generation subproblem. Letting the dual variables associated with the linear programming relaxation be $\{\pi_i\}_{i \in M}$ and μ , the subproblem is:

$$\zeta = \max\{(e - \pi)y + fw - \mu : (y, w) \in X\}$$

and *LPM* can be solved as in Section 11.3.

What happens if the solution $\bar{\lambda}$ of *LPM* is not integral? It is now not at all obvious how to recover the original variables x^k or the $\lambda_{k,t}$ variables, so the branching scheme proposed in Section 11.4 must be modified. We now consider two possibilities.

Branching Rules

(i) If $\sum_{t=1}^T \lambda_t = \alpha \notin Z$, then form two branches with $\sum_{t=1}^T \lambda_t \leq \lfloor \alpha \rfloor$ and $\sum_{t=1}^T \lambda_t \geq \lceil \alpha \rceil$ respectively.

(ii) A second possibility is based on the simple observation that if we take two elements of M , either they appear together in some subset, or not. So we choose a pair of elements (rows) i and j in M for which

$$0 < \sum_{t: y_i^t = y_j^t = 1} \lambda_t < 1,$$

and we then form two branches with $\sum_{t: y_i^t = y_j^t = 1} \lambda_t = 1$ and $\sum_{t: y_i^t = y_j^t = 1} \lambda_t = 0$ respectively.

In the first case (i) we impose that i and j lie in the same subset, and in the second case (ii) that they lie in different subsets. In case (i) all columns corresponding to subsets Q containing either i or j but not both are eliminated from the Master Problem, and the constraint $y_i = y_j$ is added to the subproblem to ensure that any new column generated does not generate a subset containing i but not j , or vice versa. In case (ii) columns containing both i and j are eliminated from the Master, and the constraint $y_i + y_j \leq 1$ is added to the subproblem.

So, imposing the constraints $y_i = y_j$ or $y_i + y_j \leq 1$ on each subproblem permits us to branch as shown in Figure 11.3.

The following result says that this second branching scheme is sufficient.

Proposition 11.3 *If $\bar{\lambda} \notin B^T$, there exist rows $i, j \in M$ such that*

$$0 < \sum_{t: y_i^t = y_j^t = 1} \lambda_t < 1$$

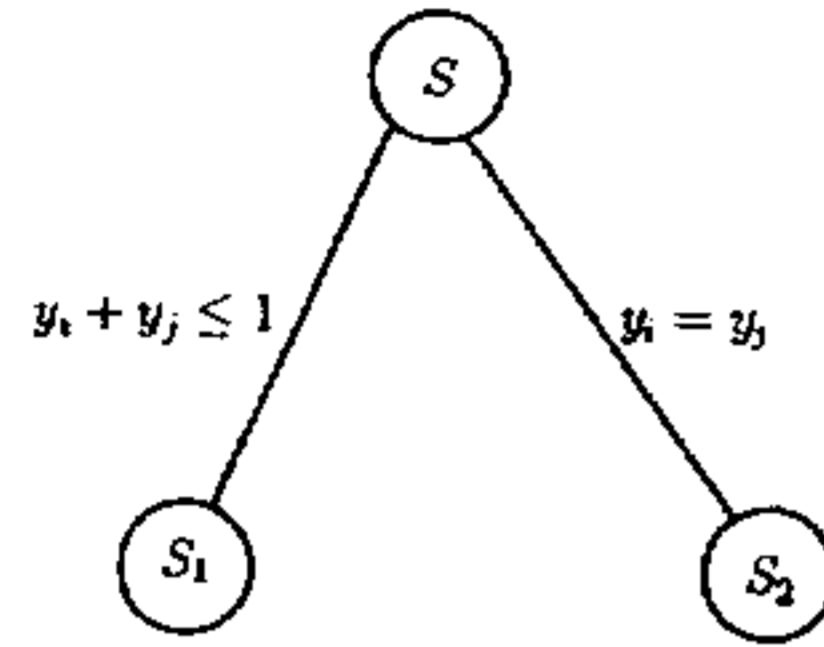


Fig. 11.3 A branching scheme for partitioning

Such a pair is also not difficult to find.

Example 11.2 Consider an instance of the clustering problem of Section 11.5 with $G = (V, E)$ the complete graph on 3 nodes, $K = 3$ clusters, the objective of choosing as many edges as possible within the clusters, and at most 2 nodes allowed per cluster, that is, node weights $d_i = 1$ for all $i \in V$, edge weights $c_e = 1$ for all $e \in E$, and cluster capacity $C = 2$.

1. *Solving LPM.* Starting the Restricted LPM with clusters consisting of single nodes leads to

$$\begin{aligned} \max & 0\lambda_1 + 0\lambda_2 + 0\lambda_3 \\ & 1\lambda_1 + 0\lambda_2 + 0\lambda_3 = 1 \\ & 0\lambda_1 + 1\lambda_2 + 0\lambda_3 = 1 \\ & 0\lambda_1 + 0\lambda_2 + 1\lambda_3 = 1 \\ & 1\lambda_1 + 1\lambda_2 + 1\lambda_3 \leq 3 \\ & \lambda \geq 0 \end{aligned}$$

with RLPM value 0, primal solution $\lambda = (1, 1, 1)$, and dual solution $\pi = (0, 0, 0), \mu = 0$. This provides a feasible solution to the original problem of objective value 0, and so we set $z = 0$.

2. *Solving the Subproblem.* The subproblem of selecting a feasible cluster of maximum reduced price is

$$\begin{aligned} \zeta = \max & w_{12} + w_{13} + w_{23} - 0y_1 - 0y_2 - 0y_3 - 0 \\ & w_{12} \leq y_1, w_{12} \leq y_2, w_{12} \geq y_1 + y_2 - 1 \\ & w_{13} \leq y_1, w_{13} \leq y_3, w_{13} \geq y_1 + y_3 - 1 \\ & w_{23} \leq y_2, w_{23} \leq y_3, w_{23} \geq y_2 + y_3 - 1 \\ & y_1 + y_2 + y_3 \leq 2 \\ & w \in \mathbb{R}^{E^+}, y \in \mathbb{R}^{V^+} \end{aligned}$$

giving $\zeta = 1$ and an optimal solution $w_{12} = y_1 = y_2 = 1$.

3. *Solution of LPM.* After three iterations LPM is solved in the form

$$\begin{aligned} \max & 0\lambda_1 + 0\lambda_2 + 0\lambda_3 + 1\lambda_4 + 1\lambda_5 + 1\lambda_6 \\ & 1\lambda_1 + 0\lambda_2 + 0\lambda_3 + 1\lambda_4 + 1\lambda_5 + 0\lambda_6 = 1 \\ & 0\lambda_1 + 1\lambda_2 + 0\lambda_3 + 1\lambda_4 + 0\lambda_5 + 1\lambda_6 = 1 \\ & 0\lambda_1 + 0\lambda_2 + 1\lambda_3 + 0\lambda_4 + 1\lambda_5 + 1\lambda_6 = 1 \\ & 1\lambda_1 + 1\lambda_2 + 1\lambda_3 + 1\lambda_4 + 1\lambda_5 + 1\lambda_6 \leq 3 \\ & \lambda \geq 0 \end{aligned}$$

with optimal solution $\lambda_4 = \lambda_5 = \lambda_6 = \frac{1}{2}$ and $z^{LPM} = \frac{3}{2}$.

4. *Branching.* Taking rows $i = 1$ and $l = 2$, we use the second branching scheme and split the problem into two subproblems:

S_1 is the set of solutions in which nodes 1 and 2 do not lie in the same cluster, so S_1 is obtained by setting $\lambda_4 = 0$ cutting off the existing solution. All new clusters containing both nodes 1 and 2 are excluded.

S_2 is the set of solutions in which any cluster containing either 1 or 2 must contain the other, so S_2 is obtained by setting $\lambda_1 = \lambda_2 = \lambda_5 = \lambda_6 = 0$ cutting off the existing solution. Any new clusters containing just one of the nodes 1,2 are excluded.

5. *Reoptimizing for S_1 .* With $\lambda_4 = 0$, the new RLPM is

$$\begin{aligned} \max & 0\lambda_1 + 0\lambda_2 + 0\lambda_3 + 1\lambda_5 + 1\lambda_6 \\ & 1\lambda_1 + 0\lambda_2 + 0\lambda_3 + 1\lambda_5 + 0\lambda_6 = 1 \\ & 0\lambda_1 + 1\lambda_2 + 0\lambda_3 + 0\lambda_5 + 1\lambda_6 = 1 \\ & 0\lambda_1 + 0\lambda_2 + 1\lambda_3 + 1\lambda_5 + 1\lambda_6 = 1 \\ & 1\lambda_1 + 1\lambda_2 + 1\lambda_3 + 1\lambda_5 + 1\lambda_6 \leq 3 \\ & \lambda \geq 0 \end{aligned}$$

with optimal primal solution $\lambda_5 = \lambda_6 = 1$, and dual solution $\pi = (0, 0, 1), \mu = 0$. The incumbent is updated, $z = 1$.

6. *Subproblem for S_1 .* The subproblem is

$$\begin{aligned} \zeta = \max & w_{12} + w_{13} + w_{23} - 0y_1 - 0y_2 - 1y_3 - 0 \\ & w_{12} \leq y_1, w_{12} \leq y_2, w_{12} \geq y_1 + y_2 - 1 \\ & w_{13} \leq y_1, w_{13} \leq y_3, w_{13} \geq y_1 + y_3 - 1 \\ & w_{23} \leq y_2, w_{23} \leq y_3, w_{23} \geq y_2 + y_3 - 1 \\ & y_1 + y_2 + y_3 \leq 2 \\ & y_1 + y_2 \leq 1 \\ & w \in \mathbb{R}^{E^+}, y \in \mathbb{R}^{V^+} \end{aligned}$$

giving $\zeta = 0$. So $LPM(S_1)$ is solved with $z^{LPM}(S_1) = 1$. The node is pruned by bound.

7. *Node S_2* . When setting $\lambda_1 = \lambda_2 = \lambda_5 = \lambda_6 = 0$, the new *RLPM* has unique optimal solution $\lambda_3 = \lambda_4 = 1$. We continue iterating between subproblem and the restricted Master till *LPM* is solved with $z^{LPM}(S_2) = 1$. Then the node is pruned by bound, and as there are no outstanding nodes, the incumbent solution $\lambda_1 = \lambda_6 = 1$ is optimal. This corresponds to one cluster containing node 1, another containing nodes 2,3, and the third necessarily empty. ■

11.7 NOTES

11.1 The fundamental paper on the decomposition of linear programs, known as Dantzig-Wolfe decomposition, is [DanWol60]. Recent surveys in this area include [Barnetal94] and [Desetal95].

11.3 The first use of column generation to solve the Master linear program arising from an integer programming problem is probably the work on the cutting stock problem [GilGom61], [GilGom63]. The equivalence of the bounds provided by the linear programming Master and the Lagrangian dual has been known since [Geo74].

11.4 The first papers on integer programming column generation appeared in the eighties [DesSouDes84], [DesSou89] on routing problems in which the subproblems are constrained shortest path problems that are solved by dynamic programming.

11.5 The multi-item lot-sizing and clustering problems have been tackled by integer programming decomposition in [Vdbeck94], and the clustering problem in [JohMehNem93], the generalized assignment problem in [Sav93], and binary and integer cutting stock problems in [Vancetal94] and [Vdbeck96] respectively.

11.6 The branching rule (ii) is from [RyaFos81]. Recent more general branching rules that are not restricted to 0-1 problems appear in [Barnetal94] and [VdbeckWol96].

In [Ben62] an alternative resource-based reformulation and decomposition approach is proposed; see Exercise 11.5.

11.8 EXERCISES

1. Consider the following instance of *UFL* with $m = 4, n = 3$,

$$(c_{ij}) = \begin{pmatrix} 2 & 1 & 5 \\ 3 & 4 & 2 \\ 6 & 4 & 1 \\ 1 & 3 & 7 \end{pmatrix} \text{ and } f = (8, 6, 5).$$

Reformulate using an Integer Programming Master Problem. Solve the Linear Programming Master by column generation.

2. Solve the following instance of *STSP* by column generation

$$(c_e) = \begin{pmatrix} - & 3 & 4 & 2 \\ - & - & 5 & 6 \\ - & - & - & 12 \\ - & - & - & - \end{pmatrix}.$$

3. Consider *GAP* with equality constraints

$$\begin{aligned} \max \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \text{ for } i = 1, \dots, m \\ \sum_{i=1}^m a_{ij} x_{ij} &\leq b_j \text{ for } j = 1, \dots, n \\ x &\in B^{mn}. \end{aligned}$$

Solve an instance with $m = 3, n = 2$, $(c_{ij}) = (a_{ij}) = \begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$, and $b =$

$\begin{pmatrix} 2 \\ 2 \end{pmatrix}$ by inter programming decomposition. Solve also by Lagrangian relaxation and by cutting planes and compare.

4. Formulate the Integer Programming Master and subproblems for the three problems presented in Section 11.5.

- 5.* (Benders' Reformulation). Use the results of Exercise 1.15 to show that

$$(MIP) \quad z = \max\{cx + hy : Ax + Gy \leq b, x \in R_+^n, y \in Y \subseteq R_+^p\}$$

has the equivalent formulation

$$\begin{aligned} z &= \max \eta \\ \eta &< u^s(b - Gy) + hy \text{ for } s = 1, \dots, S \\ u^t(b - Gy) &\geq 0 \text{ for } t = 1, \dots, T \\ y &\in Y. \end{aligned}$$