

Integer Programming

Integer Programming

Laurence A. Wolsey

UCLouvain

Second Edition

WILEY

This edition first published 2021
Copyright 2021 © by John Wiley & Sons, Inc. All rights reserved.

Edition History

John Wiley & Sons (1e, 1998)

All rights reserved. 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 or otherwise, except as permitted by law. Advice on how to obtain permission to reuse material from this title is available at <http://www.wiley.com/go/permissions>.

The right of Laurence A. Wolsey to be identified as the author of this work. has been asserted in accordance with law.

Registered Offices

John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

Editorial Office

111 River Street, Hoboken, NJ 07030, USA

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Limit of Liability/Disclaimer of Warranty

While the publisher and authors have used their best efforts in preparing this work, they make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives, written sales materials or promotional statements for this work. The fact that an organization, website, or product is referred to in this work as a citation and/or potential source of further information does not mean that the publisher and authors endorse the information or services the organization, website, or product may provide or recommendations it may make. This work is sold with the understanding that the publisher is not engaged in rendering professional services. The advice and strategies contained herein may not be suitable for your situation. You should consult with a specialist where appropriate. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Library of Congress Cataloging-in-Publication Data Applied for

ISBN: 9781119606536

Cover Design: Wiley

Cover Images: Concept of applied astronomy © Jackie Niam/Shutterstock,
LowPoly Trendy Banner © Isniper/Shutterstock

Set in 9.5/12.5pt STIXTwoText by SPi Global, Chennai, India

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To Marguerite.

Contents

Preface to the Second Edition *xii*

Preface to the First Edition *xiii*

Abbreviations and Notation *xvii*

About the Companion Website *xix*

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 15

1.8 Notes 18

1.9 Exercises 19

2 Optimality, Relaxation, and Bounds 25

2.1 Optimality and Relaxation 25

2.2 Linear Programming Relaxations 27

2.3 Combinatorial Relaxations 28

2.4 Lagrangian Relaxation 29

2.5 Duality 30

2.6 Linear Programming and Polyhedra 32

2.7 Primal Bounds: Greedy and Local Search 34

2.8 Notes 38

2.9 Exercises 38

3	Well-Solved Problems	43
3.1	Properties of Easy Problems	43
3.2	IPs with Totally Unimodular Matrices	44
3.3	Minimum Cost Network Flows	46
3.4	Special Minimum Cost Flows	48
3.4.1	Shortest Path	48
3.4.2	Maximum $s - t$ Flow	49
3.5	Optimal Trees	50
3.6	Submodularity and Matroids	54
3.7	Two Harder Network Flow Problems	57
3.8	Notes	59
3.9	Exercises	60
4	Matchings and Assignments	63
4.1	Augmenting Paths and Optimality	63
4.2	Bipartite Maximum Cardinality Matching	65
4.3	The Assignment Problem	67
4.4	Matchings in Nonbipartite Graphs	73
4.5	Notes	74
4.6	Exercises	75
5	Dynamic Programming	79
5.1	Some Motivation: Shortest Paths	79
5.2	Uncapacitated Lot-Sizing	80
5.3	An Optimal Subtree of a Tree	83
5.4	Knapsack Problems	84
5.4.1	0–1 Knapsack Problems	85
5.4.2	Integer Knapsack Problems	86
5.5	The Cutting Stock Problem	89
5.6	Notes	91
5.7	Exercises	92
6	Complexity and Problem Reductions	95
6.1	Complexity	95
6.2	Decision Problems, and Classes \mathcal{NP} and \mathcal{P}	96
6.3	Polynomial Reduction and the Class \mathcal{NPC}	98
6.4	Consequences of $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$	103
6.5	Optimization and Separation	104
6.6	The Complexity of Extended Formulations	105
6.7	Worst-Case Analysis of Heuristics	106
6.8	Notes	109
6.9	Exercises	110

7	Branch and Bound	113
7.1	Divide and Conquer	113
7.2	Implicit Enumeration	114
7.3	Branch and Bound: an Example	116
7.4	LP-Based Branch and Bound	120
7.5	Using a Branch-and-Bound/Cut System	123
7.6	Preprocessing or Presolve	129
7.7	Notes	134
7.8	Exercises	135
8	Cutting Plane Algorithms	139
8.1	Introduction	139
8.2	Some Simple Valid Inequalities	140
8.3	Valid Inequalities	143
8.4	A Priori Addition of Constraints	147
8.5	Automatic Reformulation or Cutting Plane Algorithms	149
8.6	Gomory's Fractional Cutting Plane Algorithm	150
8.7	Mixed Integer Cuts	153
8.7.1	The Basic Mixed Integer Inequality	153
8.7.2	The Mixed Integer Rounding (MIR) Inequality	155
8.7.3	The Gomory Mixed Integer Cut	155
8.7.4	Split Cuts	156
8.8	Disjunctive Inequalities and Lift-and-Project	158
8.9	Notes	161
8.10	Exercises	162
9	Strong Valid Inequalities	167
9.1	Introduction	167
9.2	Strong Inequalities	168
9.3	0–1 Knapsack Inequalities	175
9.3.1	Cover Inequalities	175
9.3.2	Strengthening Cover Inequalities	176
9.3.3	Separation for Cover Inequalities	178
9.4	Mixed 0–1 Inequalities	179
9.4.1	Flow Cover Inequalities	179
9.4.2	Separation for Flow Cover Inequalities	181
9.5	The Optimal Subtour Problem	183
9.5.1	Separation for Generalized Subtour Constraints	183
9.6	Branch-and-Cut	186
9.7	Notes	189
9.8	Exercises	190

10	Lagrangian Duality	195
10.1	Lagrangian Relaxation	195
10.2	The Strength of the Lagrangian Dual	200
10.3	Solving the Lagrangian Dual	202
10.4	Lagrangian Heuristics	205
10.5	Choosing a Lagrangian Dual	207
10.6	Notes	209
10.7	Exercises	210
11	Column (and Row) Generation Algorithms	213
11.1	Introduction	213
11.2	The Dantzig–Wolfe Reformulation of an IP	215
11.3	Solving the LP Master Problem: Column Generation	216
11.4	Solving the Master Problem: Branch-and-Price	219
11.5	Problem Variants	222
11.5.1	Handling Multiple Subproblems	222
11.5.2	Partitioning/Packing Problems with Additional Variables	223
11.5.3	Partitioning/Packing Problems with Identical Subsets	224
11.6	Computational Issues	225
11.7	Branch-Cut-and-Price: An Example	226
11.7.1	A Capacitated Vehicle Routing Problem	226
11.7.2	Solving the Subproblems	229
11.7.3	The Load Formulation	230
11.8	Notes	231
11.9	Exercises	232
12	Benders' Algorithm	235
12.1	Introduction	235
12.2	Benders' Reformulation	236
12.3	Benders' with Multiple Subproblems	240
12.4	Solving the Linear Programming Subproblems	242
12.5	Integer Subproblems: Basic Algorithms	244
12.5.1	Branching in the (x, η, y) -Space	244
12.5.2	Branching in (x, η) -Space and “No-Good” Cuts	246
12.6	Notes	247
12.7	Exercises	248
13	Primal Heuristics	251
13.1	Introduction	251
13.2	Greedy and Local Search Revisited	252
13.3	Improved Local Search Heuristics	255

13.3.1	Tabu Search	255
13.3.2	Simulated Annealing	256
13.3.3	Genetic Algorithms	257
13.4	Heuristics Inside MIP Solvers	259
13.4.1	Construction Heuristics	259
13.4.2	Improvement Heuristics	261
13.5	User-Defined MIP heuristics	262
13.6	Notes	265
13.7	Exercises	266
14	From Theory to Solutions	269
14.1	Introduction	269
14.2	Software for Solving Integer Programs	269
14.3	How Do We Find an Improved Formulation?	272
14.4	Multi-item Single Machine Lot-Sizing	277
14.5	A Multiplexer Assignment Problem	282
14.6	Integer Programming and Machine Learning	285
14.7	Notes	287
14.8	Exercises	287
References		291
Index		311

Preface to the Second Edition

There has been remarkable progress in the development of mixed integer programming solvers in the past 22 years since the first edition of this book appeared. Though branch-and-cut has replaced branch-and-bound as the basic computational tool, the underlying theory has changed relatively little and most of the cutting planes were already known at that time, but not yet implemented in the solvers. The other most significant developments in the solvers are much improved preprocessing/presolving and many new ideas for primal heuristics. The result has been a speed-up of several orders of magnitude in the codes. The other major change has been the widespread use of decomposition algorithms, in particular column generation (branch-(cut)-and-price) and Benders' decomposition.

The changes in this edition reflect these developments. Chapter 11 on column generation has been almost completely rewritten, Chapter 12 on Benders' algorithm is new and the material on preprocessing (Section 7.6), heuristics (Chapter 13), and branch-and-cut (Section 9.6) has been expanded. Other minor changes are the inclusion of a few more basic and extended formulations (for instance on fixed cost network flows in Section 3.7) and brief subsections on other topics, such as nonbipartite matching, the complexity of extended formulations or a good linear program for the implementation of lift-and-project.

Most of the exercises on formulations and algorithms are very small and typically are solved at the top node by the latest mixed integer programming (MIP) solvers. In such cases, the idea is to understand the algorithm and essentially solve the problem manually just working with a linear programming solver as a black box, or alternatively to program an algorithm for the problem. A small number of sections/subsections have a * indicating that they may be of interest, but are not essential.

I am most grateful to a variety of colleagues and friends for making suggestions and/or commenting on different chapters, including Karen Aardal, Daniele Catanzaro, Michele Conforti, Bernard Fortz, Martine Labb  , Andrea Lodi, Fabrizio Rossi, Stefano Smriglio, Eduardo Uchoa, and Dieter Weninger. Thanks again go to Fabienne Henry for her help with my latex difficulties. As before, the errors are all my own.

Preface to the First Edition

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 two to three 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 10 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 10 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 intractable 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.

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 1997–1998 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 program/programming
BM:	Benders' Master problem
BR:	Relaxation of linear program of Benders' Master 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
DSP:	LP dual of column generation or cut generation subproblem
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 program/programming
LD:	Lagrangian dual problem
lhs:	Left-hand side
LP:	Linear program/programming
LM:	Linear programming relaxation of column generation
	Master problem
$L(X)$:	Length of the input of a problem instance X
M :	A large positive number
M:	Column generation Master problem
MIP:	Mixed integer program/programming
MIR:	Mixed integer rounding

N :	Generic set $\{1, 2, \dots, n\}$
\mathcal{NP} :	Class of NP problems
\mathcal{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
RLM :	Restricted linear programming Master problem
\mathbb{R}^n :	The n -dimensional real numbers
\mathbb{R}_+^n :	The n -dimensional nonnegative real numbers
S :	Feasible region of IP, or subset of N
SOS :	Special ordered set
SP :	Column generation or cut generation separation/subproblem
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 :	Objective function of main or Master problem
\mathbb{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
$\phi(x)$:	Objective function of Benders' subproblem
ζ :	Objective function value of cut or column generation subproblem
$\mathbf{1}$:	The vector $(1, 1, \dots, 1)$
$\mathbf{0}$:	The vector $(0, 0, \dots, 0)$

About the Companion Website

This book is accompanied by a companion website:

www.wiley.com/go/wolsey/integerprogramming2e

The Instructor companion site contains the Solutions to Exercises.

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 five 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 non-linear 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/video 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. *Radiation Therapy Treatment.* In intensity modulated radiation therapy, both the shape of the beam and its intensity need to be controlled by opening and closing multileaf collimators. The goal is to provide the tumor coverage required while maintaining low radiation on critical and normal tissues.
7. *Kidney Exchange Programs.* Patients suffering from kidney failures require a transplant. Though a patient may have a donor willing to donate a kidney, an exchange between the donor–patient pair may be impossible for reasons of blood or tissue incompatibility. The problem is to organize a “best possible” set of exchanges between a number of such pairs in which a patient from one pair receives a compatible kidney from a donor in another pair.
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, VLSI, and machine learning.

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*

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

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

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

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

If all variables are integer, we have a

(Linear) Integer Program, written as

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

and if all variables are restricted to 0–1 values, we have a

0–1 or Binary Integer Program

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

Throughout the text, we will normally use the convention that if there is only one set of variables, they are denoted by x . If there are both real and integer variables, x and possibly z will denote integer variables and y and w real variables. Note that this differs from the choice of x and y in the 1st edition.

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{P} of feasible subsets of N . The problem of finding a minimum weight feasible subset can be expressed as follows:

Combinatorial Optimization Problem

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

In Section 1.3, we will see various examples of integer programs (IPs) and combinatorial optimization problems (COPs), and also see that often a COP can be formulated as an IP or a 0–1 IP.

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:

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

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

For 0–1 IPs, 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.

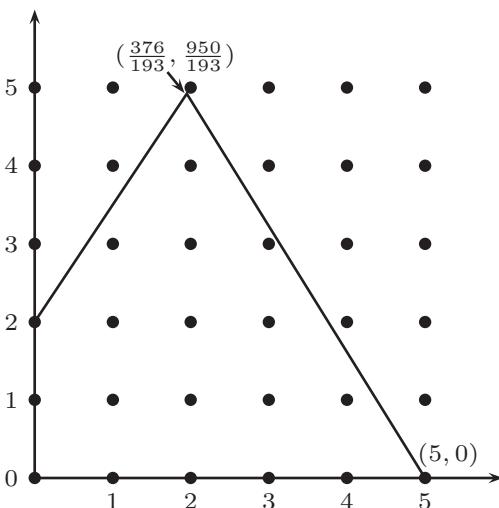


Figure 1.1 LP and IP solutions.

1.3 Formulating IPs and BIPs

As in linear programming, translating a problem description into a formulation should be done systematically. 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:

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

Each job j is done by one person:

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

The variables are 0–1:

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

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:

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

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 eight 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 0–1 IP. 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 :

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

The variables are 0–1:

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

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 deposite crates. Now, we formulate it as a 0–1 IP.

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:

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

He arrives at town j exactly once:

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

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

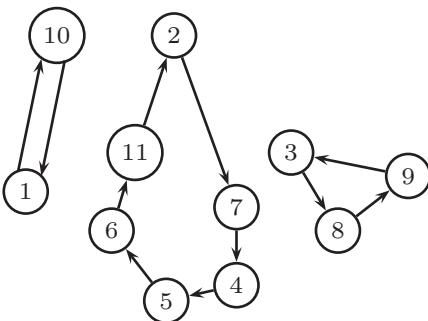


Figure 1.2 Subtours.

constraints that guarantee connectivity by imposing that the salesman must pass from one set of cities to another, so-called *cut-set* constraints:

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

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

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

The variables are 0–1:

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

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.

Table 1.1 Some typical functions.

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}

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 traveling salesman problem (TSP) with $n = 101$ has approximately 9.33×10^{157} tours.

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

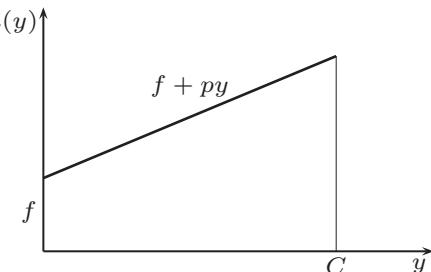
Suppose we wish to model a typical nonlinear fixed charge cost function:

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

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

Definition of an additional variable.

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

Figure 1.3 Fixed cost function.

Definition of the constraints and objective function.

We replace $h(y)$ by $fx + py$, and add the constraints $y \leq Cx, x \in \{0, 1\}$.

Note that this is not a completely satisfactory formulation, because although the costs are correct when $y > 0$, it is possible to have the solution $y = 0, x = 1$. 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 $x_j = 1$ if depot j is used, and $x_j = 0$ otherwise.

y_{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 :

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

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

$$\sum_{i \in M} y_{ij} \leq mx_j \quad \text{for } j \in N, \quad y_{ij} \geq 0 \quad \text{for } i \in M, j \in N, \quad x_j \in \{0, 1\} \quad \text{for } j \in N.$$

Definition of the objective function.

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

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

Uncapacitated Lot-Sizing (ULS)

The problem is to decide on a production plan for a single product over an n -period horizon. 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*:

y_t is the amount produced in period t .

s_t is the stock at the end of period t .

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

To handle the fixed costs, we observe that a priori no upper bound is given on y_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:

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

If we impose that $s_n = 0$, then we can tighten the variable upper bound constraints to $y_t \leq (\sum_{i=t}^n d_i) x_t$. Note also that by substituting $s_t = \sum_{i=1}^t y_i - \sum_{i=1}^t d_i$, the objective function can be rewritten as $\sum_{t=1}^n c_t y_t + \sum_{t=1}^n f_t x_t - K$ 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 $y \in \mathbb{R}^n$ satisfies $0 \leq y \leq u$, and either $a^1 y \leq b_1$ or $a^2 y \leq b_2$ (see Figure 1.4 in which the feasible region is shaded). We introduce binary variables x_i for $i = 1, 2$. Then if $M \geq \max\{a^i y - b_i : 0 \leq y \leq u\}$ for $i = 1, 2$, we take as constraints:

$$a^i y - b_i \leq M(1 - x_i) \quad \text{for } i = 1, 2$$

$$x_1 + x_2 = 1, x_i \in \{0, 1\} \quad \text{for } i = 1, 2$$

$$0 \leq y \leq u.$$

Now if $x_1 = 1$, y satisfies $a^1 y \leq b_1$, whereas $a^2 y \leq b_2$ is inactive, and conversely if $x_2 = 1$.

Such disjunctions arise naturally in scheduling problems. Suppose that two jobs must be processed on the same machine and cannot be processed 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 $t_1 \geq t_2 + p_2$.

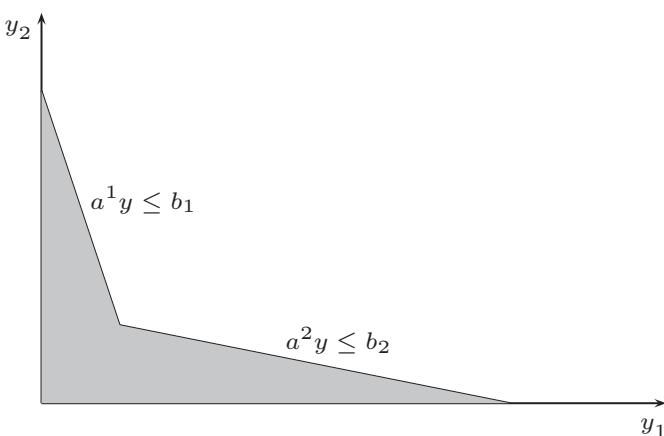


Figure 1.4 Either/or constraints.

1.6 Alternative Formulations

In Sections 1.3 and 1.5, 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 Section 1.7, 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 \mathbb{R}^n described by a finite set of linear constraints $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a *polyhedron*.

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

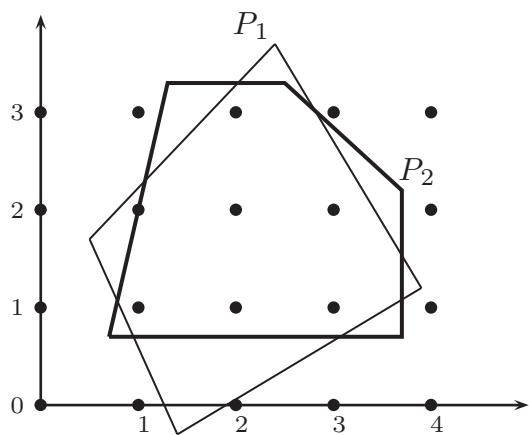
Note that this definition is restrictive. For example we saw in modeling fixed costs in Section 1.5 that we did not model the set $X = \{(0, 0), (y, 1) \text{ for } 0 < y \leq C\}$, but the set $X \cup \{(0, 1)\}$.

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

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

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 2.

Figure 1.5 Two different formulations of an IP.



Equivalent Formulations for a 0–1 Knapsack Set

Consider the set of points $X =$

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

Check that the three polyhedra below are formulations for X .

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

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

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

An Equivalent Formulation for UFL

For fixed j , consider the constraints:

$$\sum_{i \in M} y_{ij} \leq mx_j, \quad x_j \in \{0, 1\}, 0 \leq y_{ij} \leq 1 \text{ for } i \in M.$$

Logically, these constraints express the condition: if any $y_{ij} > 0$ for one or more $i \in M$, then $x_j = 1$, or stated a little differently: for each i , if $y_{ij} > 0$, then $x_j = 1$. This immediately suggests a different set of constraints:

$$0 \leq y_{ij} \leq x_j \quad \text{for } i \in M, x_j \in \{0, 1\}.$$

This leads to the alternative formulation:

$$\begin{aligned} \min & \sum_{i=1}^m \sum_{j=1}^n c_{ij} y_{ij} + \sum_{j=1}^n f_j x_j \\ & \sum_{j=1}^n y_{ij} = 1 \quad \text{for } i \in M \\ & y_{ij} - x_j \leq 0 \quad \text{for } i \in M, j \in N \\ & y_{ij} \geq 0 \text{ for } i \in M, j \in N, \quad x_j \in \{0, 1\} \text{ for } j \in N. \end{aligned}$$

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

$x_t = 1$ if production occurs in period t (as above), and $x_t = 0$ otherwise.

Definition of the constraints.

Satisfaction of demand in period t :

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

Variable upper-bound constraints:

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

The variables are mixed:

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

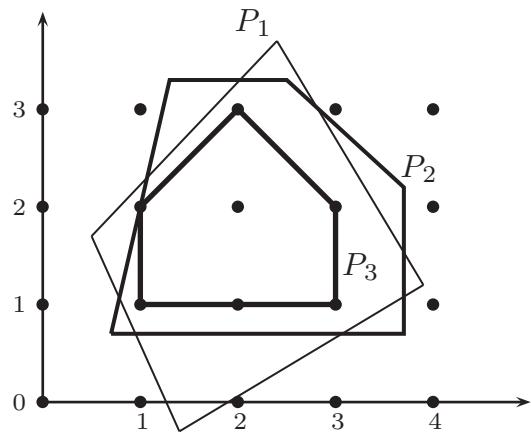
Definition of the objective function.

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

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:

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

Figure 1.6 The ideal formulation.



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 a vertex (extreme point). In this ideal case, each vertex is integer and so the IP is solved.

We can now formalize this idea.

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

Proposition 1.1 If $X = \{x \in \mathbb{Z}^n : Ax \leq b\}$, $\text{conv}(X)$ is a polyhedron.

Definition 1.4 Given a polyhedron P , x is an *extreme point* of P if $x^1, x^2 \in P$ with $x = \lambda x^1 + (1 - \lambda)x^2$, $0 < \lambda < 1$ implies that $x = x^1 = x^2$. Note that extreme points correspond to the geometric idea of vertices.

Proposition 1.2 When a linear program: $\max\{cx : x \in P\}$ where P is a polyhedron has finite optimal value, there exists an extreme point of P that is optimal.

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 holds for unbounded integer sets $X = \{x : Ax \leq b \text{ and integer}\}$, and also mixed integer sets $X = \{(x, y) : Ax + Gy \leq b, x \geq 0 \text{ and integer}, y \geq 0\}$ 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 an enormous (exponential) number of inequalities needed to describe $\text{conv}(X)$, and there is no simple characterization of all these inequalities.

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 1.5 Given a set $X \subseteq \mathbb{Z}^n$, 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 Chapter 2, 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

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

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

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

it is easily seen that $P_3 \subset P_2 \subset P_1$. 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 1.5 with a single constraint

$$\sum_{i \in M} y_{ij} \leq mx_j$$

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

$$y_{ij} \leq x_j \quad \text{for } i \in M$$

for each j . Note that if a point (x, y) satisfies the constraints $y_{ij} \leq x_j$ for $i \in M$, then summing over $i \in M$ shows that it also satisfies the constraints $\sum_i y_{ij} \leq mx_j$. 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, $y_{ij} = 1$ for $i = k(j-1) + 1, \dots, k(j-1) + k, j = 1, \dots, n$, $y_{ij} = 0$ otherwise, and $x_j = k/m$ 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

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

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

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

with $Q \subseteq \mathbb{R}^n \times \mathbb{R}^p$.

Definition 1.6 Given a polyhedron $Q \subseteq \mathbb{R}^n \times \mathbb{R}^p$, the *projection* of Q onto the subspace \mathbb{R}^n , denoted $\text{proj}_x Q$, is defined as follows:

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

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

Comparing Formulations for ULS

We can now compare the formulation P_1 :

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

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

$$\begin{aligned}s_{t-1} + y_t &= d_t + s_t \quad \text{for } t = 1, \dots, n \\ \sum_{i=1}^t w_{it} &= d_t \quad \text{for } t = 1, \dots, n \\ w_{it} &\leq d_t x_i \quad \text{for all } i, t, i \leq t \\ y_i &= \sum_{t=i}^n w_{it} \quad \text{for } i = 1, \dots, n \\ w_{it} &\geq 0 \quad \text{for all } i, t, i \leq t \\ 0 &\leq x_t \leq 1 \quad \text{for all } t.\end{aligned}$$

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 $y_t = d_t, x_t = d_t/M$ 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 later chapters on decomposition algorithms and heuristics are treated in greater detail and at a more advanced level in Nemhauser and Wolsey [233] and the excellent recent book of Conforti et al. [74]. An even more advanced theoretical treatment of integer programming is given in Schrijver [266]. Other books treating integer programming from a somewhat different angle include Martin [226] and Bertsimas and Weismantel [51]. The book of Jünger et al. [190] to celebrate 50 years of integer programming includes chapters of reminiscences from the founders of the field and up-to-date surveys of many of the major topics. For the related area of combinatorial optimization, there is the undergraduate text of Papadimitriou and Stieglitz [243] and the graduate level texts of Cook et al. [79] and of Korte and Vygen [198] and the ultimate 3-volume bible of Schrijver [267].

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*, *Transportation Science*, *OR Letters*, *Optimization Letters*, etc. contain more integer programming applications.

For those unfamiliar with linear programming, Dantzig [91] is the ultimate classic, and the book of Chvatal [71] is exceptional. Vanderbei [283] treats interior point as well as simplex methods, see also Wright [300]. For graph theory, the

books of Berge [45] and Bondy and Murty [57] are classics. For network flows Ford and Fulkerson [125] and Lawler [204] remain remarkably stimulating, while Ahuja et al. [9] is clear and comprehensive and Williamson [292] is hot off the press.

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, where possible, recent surveys and articles that can be used for further reading.

Notes for the Chapter

The importance of formulations in integer programming is well established. Certain special classes of problems that are often tackled as integer programs have had books dedicated to them: knapsack problems by Martello and Toth [224] and Kellerer et al. [195], traveling salesman problems by Applegate et al. [12], location problems by Mirchandani and Francis [230] and Laporte et al. [203], network models and network routing by Ball et al. [30, 31], vehicle routing by Golden et al. [152] and Toth and Vigo [276], and production planning by Pochet and Wolsey [250]. For machine scheduling problems, Queyranne and Schultz [254] contains a variety of integer programming formulations.

The documentation provided by some of the different mathematical programming systems, such as CPLEX [84], GUROBI [171], LINDO [210], SCIP [144, 268], and XPRESS [301], includes a wide range of 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) $u = \min\{y_1, y_2\}$, assuming that $0 \leq y_j \leq C$ for $j = 1, 2$
 - (ii) $v = |y_1 - y_2|$ with $0 \leq y_j \leq C$, for $j = 1, 2$
 - (iii) the set $X \setminus \{x^*\}$ where $X = \{x \in \{0, 1\}^n : Ax \leq b\}$ and $x^* \in X$.
3. Modeling disjunctions.

- (i) Extend the formulation of discrete alternatives of Section 1.5 to the union of two bounded polyhedra (*polytopes*) $P_k = \{y \in \mathbb{R}^n : A^k y \leq b^k, 0 \leq y \leq u\}$ for $k = 1, 2$ where $\max_k \max_i \{a_i^k y - b_i^k : 0 \leq y \leq u\} \leq M$.
- (ii) Show that an extended formulation for $P_1 \cup P_2$ is the set Q :

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

- 4.** Show that

$$\begin{aligned} X &= \{x \in \{0, 1\}^4 : 97x_1 + 32x_2 + 25x_3 + 20x_4 \leq 139\} \\ &= \{x \in \{0, 1\}^4 : 2x_1 + x_2 + x_3 + x_4 \leq 3\} \\ &= \{x \in \{0, 1\}^4 : x_1 + x_2 + x_3 \leq 2 \\ &\quad x_1 + x_2 + x_4 \leq 2 \\ &\quad x_1 + x_3 + x_4 \leq 2\}. \end{aligned}$$

- 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 1.3 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 use 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

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

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

9. Show that the set 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. Given a graph $G = (V, E)$ with edge weights $c \in \mathbb{R}^{|E|}$ and a subset $X \subseteq V$, the set of edges $E(X) = \{e = (i, j) \in E : |X \cap \{i, j\}| = 1\}$ is a *cut*. Formulate the problem of finding a maximum weight cut as
- a quadratic 0–1 integer program,
 - a linear integer program.
12. 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

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

13. 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.
14. Formulate and solve an instance of the lot-sizing problem over six 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.
15. Formulate the problem of placing N queens on an N by N chessboard such that no two queens share any row, column, or diagonal. Solve for $N = 10$.
16. Frequency Assignment. Frequencies from the range $\{1, \dots, 6\}$ must be assigned to 10 stations. For each pair of stations, there is an interference

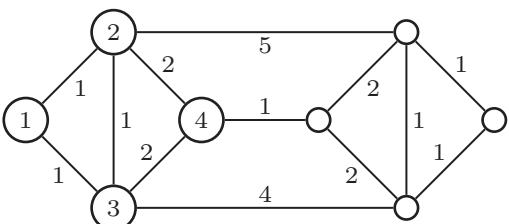


Figure 1.7 TSP Instance.

parameter which is the minimum amount by which the frequencies of the two stations must differ. The pairs with nonzero parameter are given below:

$$e = (5, 7) \quad (2, 3) \quad (3, 4) \quad (4, 5) \quad (5, 6) \quad (6, 7) \quad (7, 8) \quad (8, 9)$$

$$\quad \quad \quad 3 \quad \quad \quad 2 \quad \quad \quad 4 \quad \quad \quad 2 \quad \quad \quad 3 \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad 2$$

$$e = (9, 10) \quad (1, 8) \quad (2, 10) \quad (3, 10) \quad (5, 10) \quad (7, 10) \quad (2, 5) \quad (5, 9)$$

$$\quad \quad \quad 3 \quad \quad \quad 2 \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad 4 \quad \quad \quad 2 \quad \quad \quad 2 \quad \quad \quad 2$$

The goal is to minimize the difference between the smallest and largest frequency assigned. Formulate and solve.

2

Optimality, Relaxation, and Bounds

2.1 Optimality and Relaxation

Given an IP or COP

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

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

$$\bar{Z}_1 > \bar{Z}_2 > \cdots > \bar{Z}_s \geq Z$$

of upper bounds, and an increasing sequence

$$\underline{Z}_1 < \underline{Z}_2 < \cdots < \underline{Z}_t \leq Z,$$

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.

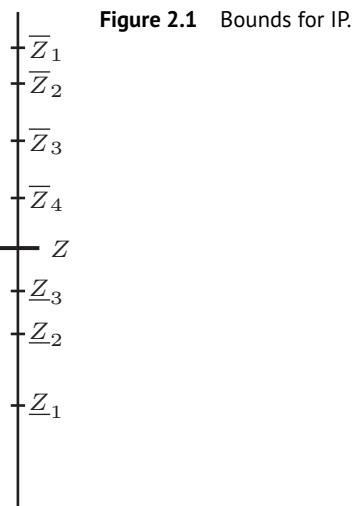
Primal Bounds

Every feasible solution $x^* \in X$ provides a lower bound $\underline{Z} = c(x^*) \leq Z$. 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

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.

Companion website: www.wiley.com/go/wolsey/integerprogramming2e



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 13.

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 on all the original feasible solutions.

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

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^{RP} and so $Z \leq f(x^*) \leq Z^{RP}$. \square

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 $Z = \max\{cx : x \in P \cap \mathbb{Z}^n\}$ with formulation $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$, the *linear programming relaxation* is the linear program $Z^{LP} = \max\{cx : x \in P\}$.

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

Example 2.1 Consider the integer program

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

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 $Z \leq 8$. \square

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 $\max\{cx : x \in X \subseteq \mathbb{Z}^n\}$ 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, $T = \emptyset$ and thus $X = \emptyset$.
- (ii) As $x^* \in X$, $Z \geq c(x^*) = f(x^*) = Z^{RP}$. As $Z \leq Z^{RP}$, we obtain $c(x^*) = Z = Z^{RP}$. \square

Example 2.2 The linear programming relaxation of the integer program:

$$\begin{aligned} \max \quad & 7x_1 + 4x_2 + 5x_3 + 2x_4 \\ \text{s.t.} \quad & 3x_1 + 3x_2 + 4x_3 + 2x_4 \leq 6 \\ & x \in \{0, 1\}^4 \end{aligned}$$

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

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,

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

- (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

$$\begin{aligned} Z^{STSP} &= \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a tour} \right\} \geq \\ Z^{1\text{-tree}} &= \min_{T \subseteq E} \left\{ \sum_{e \in T} c_e : T \text{ forms a 1-tree} \right\}. \end{aligned}$$

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

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

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

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

In Section 9.5, it will be shown how this relaxation can be solved as a series of maximum flow problems.

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

$$X' = \left\{ x \in \mathbb{Z}_+^n : \sum_{j=1}^n \lfloor a_j \rfloor x_j \leq \lfloor b \rfloor \right\},$$

where $\lfloor a \rfloor$ 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 $Z = \max\{cx : Ax \leq b, x \in X \subseteq \mathbb{Z}^n\}$. 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 $Z(u)$. \square

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 $Z = \max\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}$ and the linear program $W^{LP} = \min\{ub : uA \geq c, u \in \mathbb{R}_+^m\}$ form a weak-dual pair.

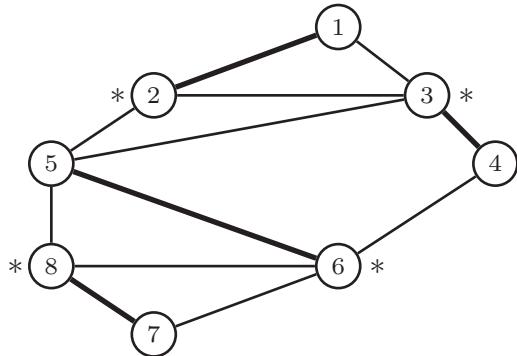
By analogy with Proposition 2.3, dual problems sometimes allow us to prove optimality.

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.

Figure 2.2 Matching and cover by nodes.



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 .

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, $|R| \geq k = |M|$. \square

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:

$$Z = \max \{\mathbf{1}x : Ax \leq \mathbf{1}, x \in \mathbb{Z}_+^m\}$$

and the minimum cardinality covering by nodes problem as follows:

$$W = \min \{\mathbf{1}y : yA \geq \mathbf{1}, y \in \mathbb{Z}_+^n\}.$$

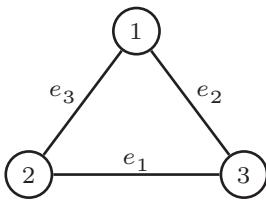


Figure 2.3 Weak duality for matching.

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.

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 $Z^{LP} = W^{LP} = 3/2$. \square

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

A Superadditive Dual. For integer programs, there is a generalization of linear programming duality, but unfortunately, it does not involve linear functions.

Definition 2.6 A function $F : \mathbb{R}^m \rightarrow \mathbb{R}$ is *superadditive* if $F(0) = 0$ and $F(u) + F(v) \leq F(u + v)$ for $u, v \in \mathbb{R}^m$.

It is *nondecreasing* if $F(u) \leq F(v)$ for $u, v \in \mathbb{R}^m$ with $u \leq v$.

One simple nondecreasing superadditive function that is not a linear function is $F^1 : \mathbb{R} \rightarrow \mathbb{R}$ with $F^1(u) = \lfloor u \rfloor$. Another is the value function of an integer program: $F^2 : \mathbb{R}^m \rightarrow \mathbb{R}$ with $F^2(d) = \max\{cx : Ax \leq d, x \in \mathbb{Z}_+^n\}$.

Theorem 2.1 *Provided it is feasible, and its linear programming relaxation has finite optimal value:*

$$\max\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\} = \min\{F(b) : F(a_j) \geq c_j \text{ for } j \in [1, n], F \in \mathcal{F}\},$$

where \mathcal{F} is the set of nondecreasing superadditive functions.

Several of the inequalities for integer programs developed in later chapters can be represented by such functions.

2.6 Linear Programming and Polyhedra

Here we remind the reader of some basic properties of linear programs and present some of the properties and alternative ways of representing polyhedra that will be used in later chapters.

Consider a linear program $\max\{cx : x \in P\}$, where $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$ with A an $m \times n$ matrix and its dual $\min\{ub : u \in U\}$ where $U = \{u \in \mathbb{R}_+^m : uA \geq c\}$. We are interested in different characterizations of feasibility and optimality.

Proposition 2.8 (Farkas' Lemma) $P \neq \emptyset$ if and only if $ub \geq 0$ for all $u \in \mathbb{R}_+^m$ such that $uA \geq 0$.

Similarly, $U \neq \emptyset$ if and only if $cx \leq 0$ for all $x \in \mathbb{R}_+^n$ such that $Ax \leq 0$.

Definition 2.7 Given a nonempty polyhedron $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$,

- (i) $x \in P$ is an *extreme point of P* if $x = x^1\lambda + x^2(1 - \lambda)$, $0 < \lambda < 1$, $x^1, x^2 \in P$ implies that $x = x^1 = x^2$. (Repeat of Definition 1.4)
- (ii) r is a *ray of P* if $r \neq 0$ and $x \in P$ implies $x + \mu r \in P$ for all $\mu \in \mathbb{R}_+^1$.
- (iii) r is an *extreme ray of P* if $r = r^1\mu_1 + r^2\mu_2$, $\mu_1, \mu_2 > 0$, with r^1, r^2 rays of P implies $r^1 = \alpha r^2$ for some $\alpha > 0$.

Note that the same rays are obtained for the cone $P^0 = \{x \in \mathbb{R}_+^n : Ax \leq 0\}$ that is always nonempty.

Theorem 2.2 (Minkowski) Every nonempty polyhedron $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$ can be represented in the form

$$P = \left\{ x \in \mathbb{R}^n : x = \sum_{s=1}^S \lambda_s x^s + \sum_{t=1}^T \mu_t v^t, \sum_{s=1}^S \lambda_s = 1, \lambda \in \mathbb{R}_+^S, \mu \in \mathbb{R}_+^T \right\},$$

where $\{x^s\}_{s=1}^S$ are the extreme points of P and $\{v^t\}_{t=1}^T$ the extreme rays of P .

Example 2.4 The polyhedron, see Figure 2.4,

$$U = \{u \in \mathbb{R}_+^2 : 4u_1 + 2u_2 \geq 2, -2u_1 + 3u_2 \geq -3, 3u_1 - u_2 \geq 1\}$$

has extreme points $(\frac{2}{5}, \frac{1}{5})^T, (\frac{1}{2}, 0)^T, (\frac{3}{2}, 0)^T$ and extreme rays $(3, 2)^T, (1, 3)^T$ giving the extended formulation:

$$U = \{u \in \mathbb{R}^2 : u = \begin{pmatrix} \frac{2}{5} \\ \frac{1}{5} \end{pmatrix} \lambda_1 + \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \lambda_2 + \begin{pmatrix} \frac{3}{2} \\ 0 \end{pmatrix} \lambda_3 + \begin{pmatrix} 3 \\ 2 \end{pmatrix} \mu_1 + \begin{pmatrix} 1 \\ 3 \end{pmatrix} \mu_2, \lambda_1 + \lambda_2 + \lambda_3 = 1, (\lambda, \mu) \in \mathbb{R}_+^3 \times \mathbb{R}_+^2\}.$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1, (\lambda, \mu) \in \mathbb{R}_+^3 \times \mathbb{R}_+^2\}.$$

□

There is also a converse stating that if $Q = \{(x, w) \in \mathbb{R}^n \times \mathbb{R}^p : Ax + Bw \leq d\}$ and $P = \text{proj}_x(Q)$, then P is a polyhedron. (See Exercise 12).

Now we return to the pair of primal and dual linear programs cited above,

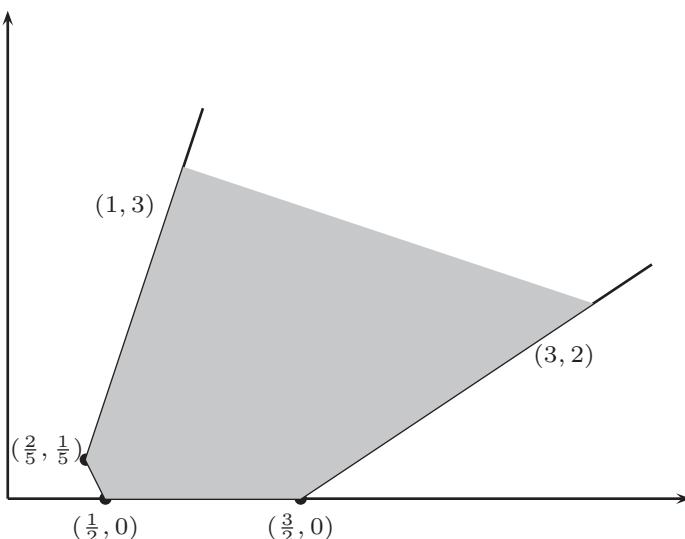


Figure 2.4 Extreme points and rays.

Proposition 2.9 Consider the linear program $Z = \max\{cx : x \in P\}$ where $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$ and its dual $W = \min\{ub : u \in U\}$ where $U = \{u \in \mathbb{R}_+^m : uA \geq c\}$.

- (i) If $P \neq \emptyset$ and $cr > 0$ for some (extreme) ray of P , then $U = \emptyset$ and $Z \rightarrow +\infty$.
- (ii) If $U \neq \emptyset$ and $vb < 0$ for some (extreme) ray of U , then $P = \emptyset$ and $W \rightarrow -\infty$.
- (iii) If $P \neq \emptyset$ and $U \neq \emptyset$, then $Z = \max_{s=1,\dots,S} cx^s = \min_{k=1,\dots,K} u^k b = W$, where $\{x^s\}_{s=1}^S$, $\{u^k\}_{k=1}^K$ are the extreme points of P and U , respectively.

2.7 Primal Bounds: Greedy and Local Search

Now, we briefly consider some simple ways to obtain feasible solutions and primal bounds. This topic is treated in considerably more detail in Chapter 13.

Heuristics from Restrictions

The converse of a relaxation is a restriction.

Definition 2.8 A problem (RE) $Z^{RE} = \max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ is a *restriction* of the integer program $Z = \max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ if

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

Proposition 2.10 If RE is a restriction of IP with optimal solution x^{RE} , x^{RE} is feasible solution to IP and $c(x^{RE}) \leq Z$.

Obvious ways to construct a restriction are to fix the values of some variables, to replace inequalities by equalities, or to add additional constraints. Typically, this is done so that the resulting restriction is an easier problem that can be solved fairly quickly.

Example 2.5 Consider the single-item lot-sizing problem with varying capacities (CLS) that can be written in the form:

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

Replacing the inequalities $y_t \leq C_t x_t$ by $y_t = C_t x_t$ and using these equations to eliminate the y_t variables leads to the restriction:

$$\begin{aligned} & \min \sum_{t=1}^n \sum_{u=1}^t (f_u + C_u c_u) x_u \\ & \sum_{u=1}^t C_u x_u \geq \sum_{u=1}^t d_u \text{ for } t = 1, \dots, n \\ & x_t \in \{0, 1\}^n \end{aligned}$$

This lot-sizing problem in which production is always at full capacity can usually be solved quickly so as to obtain a good feasible solution to CLS. \square

Greedy and Local Search Heuristics

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 2.6 The 0–1 Knapsack Problem. Consider the instance:

$$\begin{aligned} & \max 12x_1 + 8x_2 + 17x_3 + 11x_4 + 6x_5 + 2x_6 + 2x_7 \\ & 4x_1 + 3x_2 + 7x_3 + 5x_4 + 3x_5 + 2x_6 + 3x_7 \leq 9 \\ & x \in \{0, 1\}^7. \end{aligned}$$

Noting that the variables are ordered so that $\frac{c_j}{a_j} \geq \frac{c_{j+1}}{a_{j+1}}$ 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 $Z^G = cx^G = 22$. \square

Example 2.7 The Symmetric Traveling Salesman Problem. Consider an instance with cost/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 cost.

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 cost $\sum_e c_e x_e = 54$. \square

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 2.8 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

$$Q(S^3) = \{\{1, 3\}, \{2, 3\}, \{3, 4\}, \emptyset\}.$$

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

Example 2.9 The Graph Equipartition Problem. Given a graph $G = (V, E)$ and $n = |V|$, the problem is to find a subset of nodes $S \subset V$ 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. \square

2.8 Notes

- 2.1 The concept of a relaxation, and the complementary idea of a are formalized in Geoffrion and Marsten [139].
- 2.2 Linear programming relaxations have been present ever since combinatorial problems were first formulated as linear integer programs, see Dantzig et al. [93] and Dantzig [90].
- 2.3 The assignment relaxation for the traveling salesman problem was already used in Little et al. [211], and the 1-tree relaxation was introduced in Held and Karp [174].
- 2.4 Chapter 10 is on Lagrangian relaxation. Other relaxations studied include group or modular relaxations, see Gomory [156] and also Ch. II.3 in Nemhauser and Wolsey [233] and surrogate relaxations Glover [145].
- 2.5 A strong duality for the general matching problem appears in the classic paper of Edmonds [102], which has had a major influence on the development of combinatorial optimization. See also Section 4.4. Several of the most beautiful results in this field are strong duality theorems, see Cook et al. [79] and Korte and Vygen [198]. For integer programs, a general superadditive duality theory was developed in the 1970s based on the work of Gomory [157] and Gomory and Johnson [159].
- 2.6 The converse of Theorem 2.10 of Minkowski [228] is due to Weyl [291]. Some further properties of polyhedra are developed in Section 9.2.
- 2.7 The greedy and local exchange heuristics are formalized in Chapter 13. Other metaheuristics such as simulated annealing and tabu search are also introduced, as well as heuristics for finding good feasible solutions with an MIP solver.

2.9 Exercises

1. Find a maximum cardinality matching in the graph of Figure 2.5a by inspection. Give a proof that the solution found is optimal.

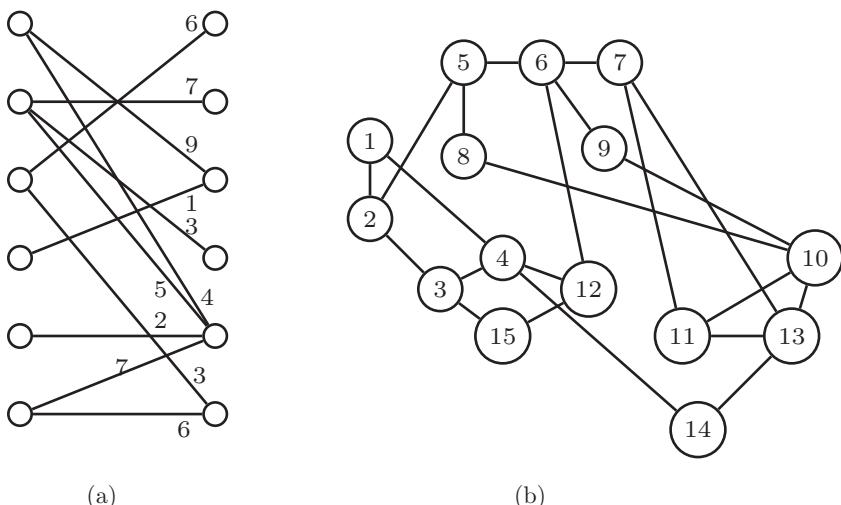


Figure 2.5 (a) Matching instance. (b) Stable set instance.

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 2.5b.
3. Find primal and dual bounds for the integer knapsack problem:

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

4. Consider the 0–1 integer program:

$$(P_1) \quad \max \left\{ cx : \sum_{j=1}^n a_{ij}x_j = b_i \text{ for } i = 1, \dots, m, x \in \{0, 1\}^n \right\},$$

and the 0–1 equality knapsack problem

$$(P_2), \quad \max \left\{ cx : \sum_{j=1}^n \left(\sum_{i=1}^m u_i a_{ij} \right) x_j = \sum_{i=1}^m u_i b_i, x \in \{0, 1\}^n \right\},$$

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

5. Consider the equality integer knapsack problem:

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

(i) Show that the problem

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

is a relaxation of P_1 .

(ii) Show that the problem

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

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

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

is a strong dual problem.

7. Network formulation of the integer knapsack problem with positive integer coefficients $a_j > 0$ for $j = 1, \dots, n$ and $b > 0$. Let

$$G(d) = \max \left\{ cx : \sum_{j=1}^n a_j x_j \leq b, x \in \mathbb{Z}_+^n \right\}$$

denoted problem $P(d)$.

(i) Show that $G(d) \geq G(d - a_j) \forall a_j \leq d \leq b$ and $j = 1, \dots, n$ and $G(d) \geq G(d - 1) \forall 1 \leq d \leq b$.

(ii) Show that

$$\min y(b)$$

$$y(d) - y(d - a_j) \geq c_j \quad a_j \leq d \leq b, j = 1, \dots, n$$

$$y(d) - y(d - 1) \geq 0 \quad 1 \leq d \leq b$$

$$y(0) = 0$$

is a strong dual of the knapsack problem.

(iii) Take the dual of this linear program and interpret the variables and constraints of the resulting LP.

8. Formulate the maximum cardinality matching problem of Figure 2.5(a) 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.
9. (i) Show that the value function of an IP: $\phi(d) = \max\{cx : Ax \leq d, x \in \mathbb{Z}_+^n\}$ is superadditive and nondecreasing.
(ii) Prove the Strong Duality Theorem 2.1.
10. Use LP duality to demonstrate Farkas' Lemma.
11. *Fourier–Motzkin Elimination.* Let Q be the polyhedron

$$\begin{aligned} a^i y + w &\leq b_i \quad i = 1, \dots, m \\ c^j y - w &\leq d_j \quad j = 1, \dots, n \\ e^k y &\leq f_k \quad k = 1, \dots, K \\ y \in \mathbb{R}^p, w \in \mathbb{R}^1. \end{aligned}$$

Show that $P = \text{proj}_y(Q)$ is the polyhedron

$$\begin{aligned} (a^i + c^j)y &\leq b_i + d_j \quad i = 1, \dots, m, j = 1, \dots, n \\ e^k y &\leq f_k \quad k = 1, \dots, K \\ y \in \mathbb{R}^p. \end{aligned}$$

Repeating this procedure allows one to project any polyhedron onto a subset of its variables. Note that the number of constraints in the projected polyhedron can increase exponentially.

12. (Weyl's Theorem). Show that the projection of a polyhedron is again a polyhedron.
13. Show that if $X \subset \{0, 1\}^n$, every point in X is an extreme point of $\text{conv}(X)$.
14. Apply a greedy heuristic to the instance of the uncapacitated facility location problem in Example 2.8.
15. Define greedy and local search heuristics for the maximum cardinality stable set problem discussed in Exercise 2.
16. Devise a greedy heuristic for the set covering problem.

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 $\max\{cx : x \in X \subseteq \mathbb{R}^n\}$, 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 $x^* \in \mathbb{R}^n$, 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) $\max\{cx : x \in X \subseteq \mathbb{R}^n\}$, there exists an efficient (polynomial) algorithm.
- (ii) *Strong Dual Property*: For the given problem class, there exists a strong dual problem (D) $\min\{\omega(u) : u \in U\}$ 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 :

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

with integral data (A, b) is to ask when we will be lucky, and the linear programming (LP) relaxation $\max\{cx : Ax \leq b, x \in \mathbb{R}_+^n\}$ 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 . \square

The next step is to ask when we 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 .

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

Table 3.1 Matrices that are not TU.

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

Table 3.2 Matrices that are TU.

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

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. \square

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 $\max\{cx : Ax \leq b, x \in \mathbb{R}_+^n\}$ 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: $\max\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}$ 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 $\text{conv}(X) = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ 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:

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

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

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

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 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: $0 \leq x_{ij} \leq h_{ij}$. \square

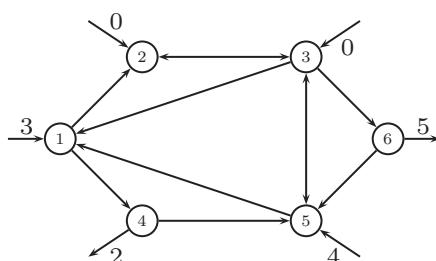
Proposition 3.4 *The constraint matrix A arising in a minimum cost network flow problem is totally unimodular.*

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 $M_2 = \emptyset$. \square

Corollary 3.1 *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) and (3.3) describe the convex hull of the integral feasible flows.*

Figure 3.1 Digraph for minimum cost network flow.



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 follows:

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

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

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

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

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

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

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

Theorem 3.1 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

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

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. \square

We note that one optimal 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 follows:

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

The dual is

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

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

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

where the first inequality follows from $h \geq 0, w \geq 0$ and the second 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 3.2 *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 3.5 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.

As in the greedy heuristic in Section 2.7, 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

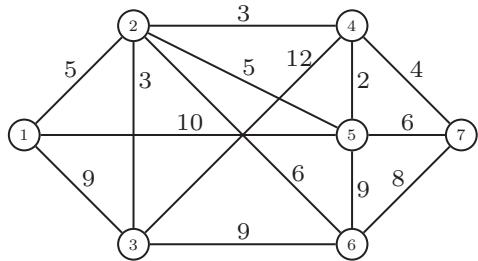
Initialization. Set $E^0 = E$, $T^0 = \emptyset$. Order the edges by nonincreasing weight $c_1 \geq c_2 \geq \dots \geq c_m$, where c_t is the cost of edge e_t .

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 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 $c_{t+1} \leq 0$.

Figure 3.2 Graph for optimal weight tree.



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

Theorem 3.3 *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 3.5 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 1.3 we saw how to avoid cycles. Thus, the maximum weight forest problem can be formulated as follows: ($E(S) = \{e = (i, j) \in E : i \in S, j \in S\}$).

$$\max \sum_{e \in E} c_e x_e \quad (3.10)$$

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

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

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

Theorem 3.4 *The convex hull of the incidence vectors of the forests in a graph is given by the constraints (3.11) and (3.12).*

The incidence vectors of the spanning trees are obtained by setting (3.11) with $S = V$ as an equality. It follows that

Corollary 3.2 *The convex hull of the incidence vectors of the spanning trees in a graph is given by the constraints (3.11) and (3.12) and the equation*

$$\sum_{e \in E} x_e = n - 1. \quad (3.14)$$

Theorem 3.4 shows that the Explicit Convex Hull Property holds for the Maximum Weight Forest Problem. We generalize and prove this result in the next section, and later in Section 9.5, we show that the Efficient Separation Property holds for this problem.

The formulation (3.11), (3.12), and (3.14) has an exponential number of constraints. Is there a formulation for the convex hull of the incidence vectors of the spanning trees with only a polynomial number of constraints and variables?

In fact, there are numerous formulations for this set. Here, we present two that give two different viewpoints on spanning trees, one based on directed cuts and the other on flows.

A Directed Flow and a Directed Cut Formulation

If we replace each edge of the graph by two directed arcs, we obtain a digraph $D = (V, A)$. Selecting a root node r , the subdigraph obtained from a tree contains a directed path from r to every node $k \in V \setminus \{r\}$. So one unit of flow can be sent from r to each node k .

Define variables $z_{ij} = 1$ if arc (i, j) lies on one of these directed paths and w_{ij}^k if the unit of flow destined for node k uses the arc (i, j) . This leads to the extended formulation (Q_{flow}):

$$\begin{aligned} & - \sum_{j \in V \setminus \{r\}} w_{rj}^k = -1 \quad \text{for } k \in V \setminus \{r\} \\ & \sum_{i \in V \setminus \{j\}} w_{ij}^k - \sum_{i \in V \setminus \{j\}} w_{ji}^k = 0 \quad \text{for } j, k \in V \setminus \{r\}, j \neq k \\ & \sum_{i \in V \setminus \{k\}} w_{ik}^k = 1 \quad \text{for } k \in V \setminus \{r\} \\ & w_{ij}^k \leq z_{ij} \quad \text{for } (i, j) \in A, k \in V \setminus \{r\} \\ & \sum_{(i, j) \in A} z_{ij} = n - 1 \\ & x_e = z_{ij} + z_{ji} \quad \text{for } e \in E \\ & \sum_{j \in V \setminus \{r\}} z_{jr} = 0 \\ & w \in \mathbb{R}_+^{|A|(|V|-1)}, z \in \mathbb{R}_+^{|A|}, x \in [0, 1]^{|E|}. \end{aligned}$$

Now, by the max flow min cut Theorem 3.2, there is such a flow if and only if the capacity of every cut set separating r from some nodes of $V \setminus \{r\}$ is at least 1 unit. This leads to the extended formulation (Q_{dicut}):

$$\begin{aligned} & \sum_{(i, j) \in \delta^-(S)} z_{ij} \geq 1 \quad \text{for } S \subseteq V \setminus \{r\}, S \neq \emptyset \\ & \sum_{(i, j) \in A} z_{ij} = n - 1 \\ & x_e = z_{ij} + z_{ji} \quad \text{for } e \in E \\ & \sum_{j \in V \setminus \{r\}} z_{jr} = 0 \\ & z_{ij} \in \mathbb{R}_+^{|A|}, x \in [0, 1]^{|E|}. \end{aligned}$$

Theorem 3.5 Both $\text{proj}_x(Q_{\text{flow}})$ and $\text{proj}_x(Q_{\text{dicut}})$ give the convex hull of the incidence vectors of the spanning trees in a graph.

See Exercise 15 for a proof based on Theorem 3.4.

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 $f : \mathcal{P}(N) \rightarrow \mathbb{R}^1$ is *submodular* if

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

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

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

An alternative representation of such functions is useful.

Proposition 3.6 *A set function f is nondecreasing and submodular if and only if*

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

Proof. Suppose f is nondecreasing 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 nondecreasing and the second from submodularity. The other direction is immediate. \square

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

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

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 3.6 *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:

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

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 $j \leq r$, $\sum_{S: j \in S} y_S \geq \sum_{i=j}^r y_{S^i} = \sum_{i=j}^{r-1} (c_i - c_{i+1}) + c_r = c_j$, 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. \square

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 3.7 *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 3.6 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 $x^A \in P(r)$. \square

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 3.6, we know how to optimize on matroids.

Theorem 3.7 *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 $r : \mathcal{P}(E) \rightarrow \mathbb{R}^1$, 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) and (3.12) is a special case of a submodular polyhedron.

The constraint set associated with 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 $c \in \mathbb{Z}^n$ 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 3.6 that the linear system $\left\{ \sum_{j \in S} x_j \leq f(S) \text{ for } S \subseteq N, x_j \geq 0 \text{ for } j \in N \right\}$ 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 3.8 If $Ax \leq b$ is TDI, b is an integer vector, and $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ 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 Two Harder Network Flow Problems*

Here we present two problems that are important generalizations of the maximum/minimum weight spanning tree problem and the minimum cost flow problem, respectively.

The Steiner Tree Problem

Given a graph $G = (V, E)$ and a set $T \subseteq V$ of terminals, a *Steiner tree* is a tree connecting all the nodes in T . Note that the Steiner tree may or may not contain each node in $V \setminus T$. Given weights $c \in \mathbb{R}_+^{|E|}$ and the set T , the problem is to find a minimum weight Steiner tree.

A first formulation involves the selection of a root node $r \in T$ and cut sets.

Let $x_e = 1$ if edge e is part of the Steiner tree. There must be a path between node r and nodes in $T \setminus \{r\}$ that in turn must intersect a cut. This leads to

$$\begin{aligned} \min & \sum_{e \in E} c_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq 1 \quad \text{for } S \subset V, r \in S, T \setminus S \neq \emptyset \\ & x \in \{0, 1\}^{|V|} \end{aligned}$$

Unfortunately, this formulation is fairly weak. One way to strengthen it is again to pass to a directed version. Having chosen the root $r \in T$, let $z_{ij} = 1$ if the arc i, j lies on the directed path from r to a node in $T \setminus \{r\}$. Now we obtain:

$$\begin{aligned} \min & \sum_{e \in E} c_e x_e \\ & \sum_{(i,j) \in \delta^+(S)} z_{ij} \geq 1 \quad \text{for } S \subseteq V, r \in S, T \setminus S \neq \emptyset \end{aligned} \tag{3.15}$$

$$z_{ij} + z_{ji} = x_e \quad \text{for } e = (i, j) \in E \tag{3.16}$$

$$z \in \{0, 1\}^{|A|}, x \in \{0, 1\}^{|E|}. \tag{3.17}$$

The associated formulation is tighter than the undirected formulation, but is not always integral. See Exercise 16.

Observe that when $T = V$, this is the optimal tree problem, and when $|T| = 2$, it is the shortest path problem.

The Fixed Charge Network Flow Problem

Given a directed graph (network) $D = (V, A)$, arc weights $c \in \mathbb{R}_+^{|A|}$, unit flow costs $f \in \mathbb{R}_+^{|A|}$ and demands/supplies $b \in \mathbb{R}^{|V|}$, where $b_i > 0$ denotes a demand and $b_i < 0$ a supply at node i . For feasibility $\sum_{i \in V} b_i = 0$. The problem is to find a feasible flow minimizing the sum of the arc costs plus the unit flow costs. Let $T^+ = \{i : b_i > 0\}$ be the set of demand (or sink) nodes and $T^- = \{i : b_i < 0\}$ the set of supply (or source) nodes. B is the maximum flow in an arc. In the uncapacitated case, we can take $B = \sum_{i \in T^+} b_i$. Taking $x_{ij} = 1$ if arc (i, j) is used and y_{ij} to be the flow in arc (i, j) , a natural formulation is

$$\min \sum_{(i,j) \in A} (c_{ij}x_{ij} + f_{ij}y_{ij}) \quad (3.18)$$

$$\sum_{j \in V \setminus \{i\}} y_{ji} - \sum_{j \in V \setminus \{i\}} y_{ij} = b_i \quad \text{for } i \in V \quad (3.19)$$

$$y_{ij} \leq Bx_{ij} \quad \text{for } (i, j) \in A \quad (3.20)$$

$$y_{ij} \in \mathbb{R}_+^{|A|}, x \in \{0, 1\}^{|A|}. \quad (3.21)$$

Again, a useful idea for obtaining a tighter formulation is to consider the flow going to each node in T^+ as a separate commodity. So we define w_{ij}^k to be the flow in arc (i, j) with destination node $k \in T^+$. This leads to the extended formulation:

$$\begin{aligned} & \min \sum_{(i,j) \in A} (c_{ij}x_{ij} + f_{ij}y_{ij}) \\ & \sum_{j \in V \setminus \{i\}} w_{ji}^k - \sum_{j \in V \setminus \{i\}} w_{ji}^k = \beta_i^k \quad \text{for } i \in T^-, k \in T^+ \\ & \sum_{j \in V \setminus \{i\}} w_{ji}^k - \sum_{j \in V \setminus \{i\}} w_{ji}^k = 0 \quad \text{for } i \in V \setminus (T^+ \cup T^-), k \in T^+ \\ & \sum_{j \in V \setminus \{i\}} w_{ji}^k - \sum_{j \in V \setminus \{i\}} w_{ji}^k = b_k \quad \text{for } i = k, k \in T^+ \\ & w_{ij}^k \leq b_k x_{ij} \quad \text{for } k \in T^+, (i, j) \in A \\ & \sum_{k \in T^+} w_{ij}^k = y_{ij} \quad \text{for } (i, j) \in A \\ & \sum_{i \in T^-} \beta_i^k = b_k \quad \text{for } k \in T^+ \end{aligned}$$

$$w \in \mathbb{R}_+^{|A| \cdot |T^+|}, y \in \mathbb{R}_+^{|A|}, x \in \{0, 1\}^{|A|}.$$

Note that the extended formulation for the uncapacitated lot-sizing problem presented in Section 1.6 is of this type with $|T^-| = 1$, i.e just a single source node. In this very special case, the tightened multicommodity formulation gives the convex hull of solutions.

3.8 Notes

- 3.1 The theoretical importance of the separation problem is discussed at the end of Chapter 6. Its solution provides a way to generate strong valid inequalities as cutting planes, see Chapter 9.
- 3.2 Totally unimodular matrices have been studied since the 1950s. The characterization of Proposition 3.1 is due to Hoffman and Kruskal [179]. The interval or consecutive 1's property of Exercise 3 is due to Fulkerson and Gross [128], and the stronger necessary condition is from Ghoul-Houri [140]. A complete characterization of TU matrices is much more difficult; see Seymour [269] or the presentation in Schrijver [266].
- 3.3 We again refer to Ahuja et al. [9] for network flows, as well as shortest path and max flow problems. The former book also contains a large number of applications and a wealth of exercises. See also Williamson [292].
- 3.4 The max flow min cut theorem was already part of the max flow algorithm of Ford and Fulkerson [124]. 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 Gomory and Hu [158]. For an algorithm that find minimum cuts directly without using flows, see Ch. 3 in Cook et al. [79].
- 3.5 The greedy algorithm for finding minimum weight trees is from Kruskal [199]. A faster classical algorithm is that of Prim [251]. The $O(n^3)$ extended formulation for the spanning tree polytope is due to Wong [299]. An alternative $O(n^3)$ extended formulation was given by Martin [225].
- 3.6 Submodular polyhedra and the greedy algorithm for matroids are studied in Edmonds [103, 104], see also Lawler [204]. Volume B of Schrijver [267] covers the topic in depth. For total dual integrality, see Edmonds and Giles [105].
- 3.7 The multicommodity extended formulation for fixed charge network flows was proposed by Rardin and Choe [257]. Magnanti and Wolsey [217] and Goemans [149] contain a discussion of alternative formulations for tree and Steiner tree problems, respectively.

3.9 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 $P = \{(x, y_1, \dots, y_m) \in \mathbb{R}_+^{m+1} : x \leq 1, y_i \leq x \text{ for } i = 1, \dots, m\}$ 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

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

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 3.3a. Use an optimal dual solution to prove that your solution is optimal.

7. Use linear programming to find a minimum $s - t$ cut in the capacitated network of Figure 3.3b.

8. Find a minimum weight spanning tree in the graph shown in Figure 3.4a.

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

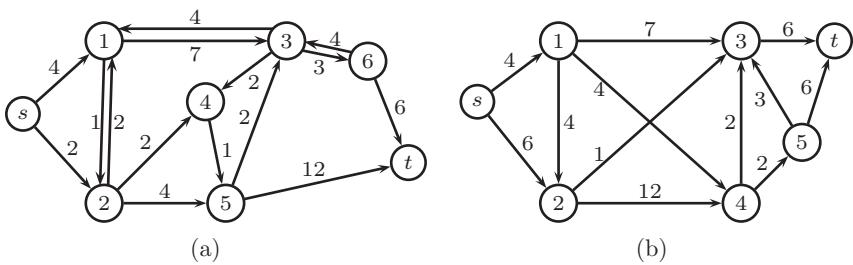


Figure 3.3 (a) Shortest path instance. (b) Network instance.

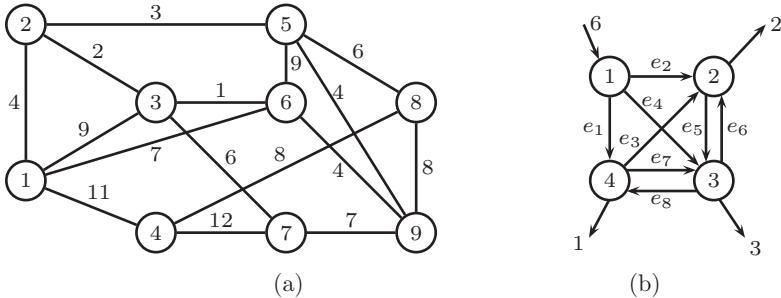


Figure 3.4 (a) Tree instance, (b) fixed charge network flow instance.

10. Consider a single source fixed charge network instance with four nodes and node 1 as the source, see Figure 3.4b, The demands at each node are $b = (-6, 2, 3, 1)$. There are eight arcs with tails and heads given by $t = (1, 1, 4, 1, 2, 3, 4, 3)$ and $h = (4, 2, 2, 3, 3, 2, 3, 4)$, respectively. The capacity of each arc is 6 and the fixed costs on the arcs are $c = (5, 2, 7, 3, 2, 4, 9, 12)$. Solve using the basic formulation (3.18)–(3.21).
11. Show that a set function f is submodular on N if and only if $\rho_j(S) \geq \rho_j(T)$ for all $S \subseteq T$, where $\rho_j(S) = f(S \cup j) - f(S)$. Show that it is nondecreasing if and only if $\rho_j(S) \geq 0$ for all $S \subseteq N$. See Definition 3.5.
12. Show that the 0-1 covering problem

$$\begin{aligned} z &= \min \sum_{j=1}^n f_j x_j \\ &\sum_{j=1}^n a_{ij} x_j \geq b_i \quad \text{for } i \in M \\ x &\in \{0, 1\}^n \end{aligned}$$

with $a_{ij} \geq 0$ for $i \in M, j \in N$ can be written in the form

$$z = \min \left\{ \sum_{j \in S} f_j : g(S) = g(N) \right\},$$

where $g : \mathcal{P}(N) \rightarrow R_+^1$ is a nondecreasing set function.

- (i) What is g ?
- (ii) Show that g is a submodular set function.

13. 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 ?
14. 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 $B \cup \{j\}$ is independent, and
 - (ii) for an arbitrary set $A \subseteq N$, every maximal independent set in A has the same cardinality.
15. Prove Theorem 3.5 for Q_{dicut} . Specifically:
 Show that if $(x, z) \in Q_{\text{dicut}}$, then
 - (i) $z_{jr} = 0$ for all $j \in V \setminus \{r\}$.
 - (ii) $\sum_{i \in V \setminus \{j\}} z_{ij} = 1$ for all $j \in V \setminus \{r\}$.
 - (iii) $\sum_{(i,j) \in A(S)} z_{ij} \leq |S| - 1$ for all $S \subseteq V$
 - (iv) x satisfies (3.11) and (3.12) and (3.14).
 The opposite direction is simple.
16. Consider the Steiner tree problem on the graph of Figure 3.5 with terminal nodes 0, 4, 5, 6 and a weight of 1 on all the edges. Show that the LP relaxation of the directed cut formulation (3.15)–(3.17) is not integral by describing a fractional LP solution of smaller value than that of the optimal integer solution.

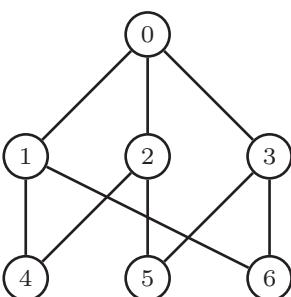


Figure 3.5 Steiner tree instance.

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 linear programming 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* $\max\{|M| : M \text{ is a matching}\}$. 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 .

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.

Companion website: www.wiley.com/go/wolsey/integerprogramming2e

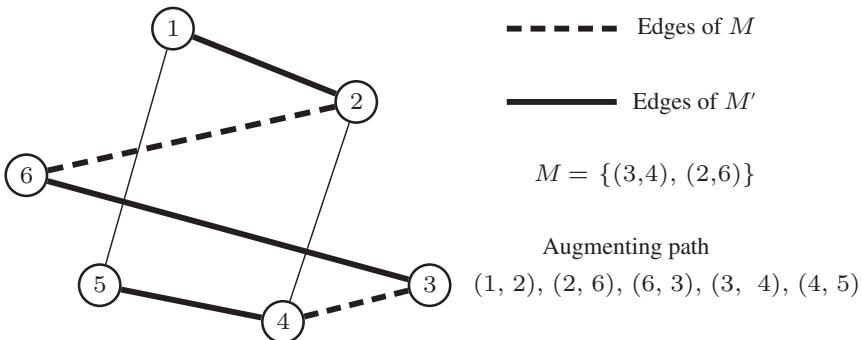


Figure 4.1 An augmenting path.

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, $|M'| = |M| + 1$. \square

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. \square

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 Section 4.2, 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 4.1 *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 $|R| = |M|$. \square

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 $(4, 10), (5, 10), (5, 9)$.

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. \square

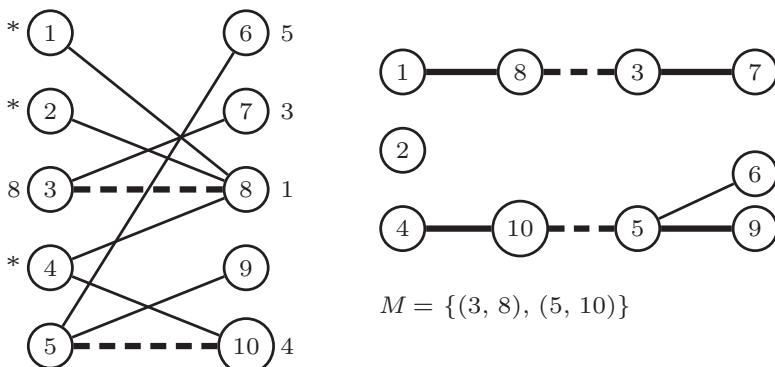


Figure 4.2 Bipartite matching.

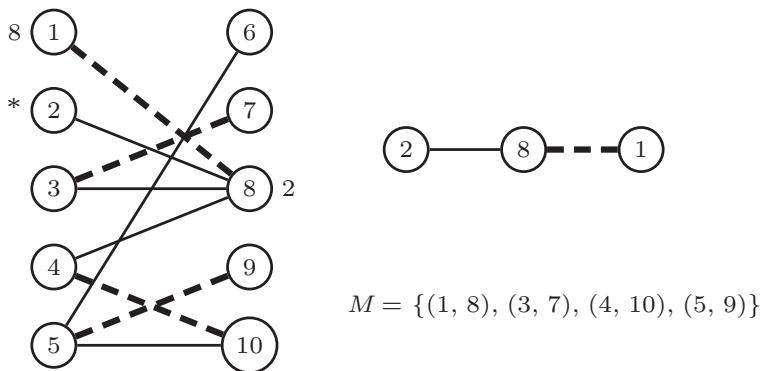


Figure 4.3 Bipartite matching 2.

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:

$$\begin{aligned} Z &= \max \sum_{e \in E} c_e x_e \\ \sum_{e \in \delta(i)} x_e &\leq 1 \text{ for } i \in V_1 \cup V_2 \\ x_e &\geq 0 \text{ for } e \in E. \end{aligned}$$

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:

$$\begin{aligned} Z &= \max \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \\ \sum_{j=1}^n x_{ij} &= 1 \text{ for } i = 1, \dots, n \\ \sum_{i=1}^n x_{ij} &= 1 \text{ for } j = 1, \dots, n \\ x_{ij} &\geq 0 \text{ for } i, j = 1, \dots, n. \end{aligned}$$

Below we develop an algorithm for the assignment problem. Afterward, 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:

$$\begin{aligned} W &= \min \sum_{i=1}^n u_i + \sum_{j=1}^n v_j \\ u_i + v_j &\geq c_{ij} \text{ for } i, j = 1, \dots, n. \end{aligned}$$

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 4.3 *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 is always the constant $\sum_{i=1}^n u_i + \sum_{j=1}^n v_j$, and the claim follows. \square

Now, we wish to characterize an optimal solution.

Proposition 4.4 *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 4.3, x is also optimal with weights c_{ij} and has value $\sum_{i=1}^n u_i + \sum_{j=1}^n v_j$. \square

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 4.4, 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 Section 4.2.

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 Section 4.2 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:

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

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 4.5 *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 $*$. \square

Proposition 4.6 *In the dual step, $\delta > 0$.*

Proof. We observed in the proof of Theorem 4.1 that there are no edges of \bar{E} between V_1^+ and $V_2 \setminus V_2^+$. Therefore, $\bar{c}_{ij} < 0$ for all $i \in V_1^+, j \in V_2 \setminus V_2^+$. \square

Proposition 4.7 *After a dual change,*

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

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

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

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

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 $\bar{c}_{ij} = 0$. \square

Proposition 4.8 *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. \square

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

Proof. By Observations 4.2–4.4, 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 $O(|E|)$. \square

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

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

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

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

and a dual objective value of 64.

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.

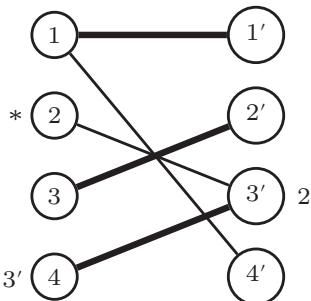


Figure 4.4 Primal step.

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

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

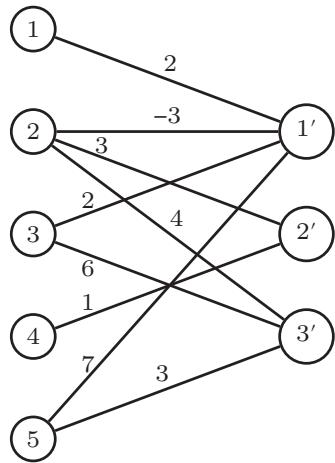
The dual solution now has value 62.

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. \square

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.

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

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

Figure 4.5 Maximum weight bipartite matching.

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 Matchings in Nonbipartite Graphs*

When the graph $G = (V, E)$ is nonbipartite, we saw in Section 2.5 that the obvious integer programming formulation for the maximum weight matching problem:

$$\begin{aligned} Z = \max \sum_{e \in E} c_e x_e \\ \sum_{e \in \delta(i)} x_e \leq 1 \text{ for } i \in V, \end{aligned} \tag{4.1}$$

$$x \in \mathbb{Z}_+^{|E|}, \tag{4.2}$$

does not provide the convex hull of the set X^M of incidence vectors of the matchings when G is a triangle.

For the triangle with edges e_1, e_2, e_3 , it is clear that a matching can contain at most one edge, so all the corresponding incidence vectors satisfy

$$x_{e_1} + x_{e_2} + x_{e_3} \leq 1.$$

More generally, as any edge is incident to two vertices, the *blossom inequality*

$$\sum_{e \in E(S)} x_e \leq \lfloor \frac{|S|}{2} \rfloor, \tag{4.3}$$

is valid for any $S \in \mathcal{S} = \{S \subseteq V : |S| \text{ is odd}\}$.

One of the most beautiful and far-reaching results that launched the field of combinatorial optimization is the following:

Theorem 4.2 $\text{conv}(X^M)$ is completely described by the degree constraints (4.1), the blossom inequalities (4.3), and the nonnegativity constraints.

What is more, there is a fast combinatorial algorithm based on augmenting paths and a “shrinking operation for odd sets” to find a maximum weight matching.

Another interesting aspect of the algorithm is that the linear programming dual

$$\begin{aligned} \min \sum_{i \in V} u_i + \sum_{S \in S} \left\lfloor \frac{|S|}{2} \right\rfloor y_S \\ u_i + u_j + \sum_{S \in S : (i,j) \in E(S)} y_S \geq c_e \text{ for } e = (i,j) \in E, \\ u \in \mathbb{R}_+^{|V|}, y_S \geq 0 \text{ for } S \in S, \end{aligned}$$

has an integer optimal solution when $c \in \mathbb{Z}^{|E|}$.

In particular, taking $c_e = 1$ for all $e \in E$ gives a strong dual for the size of a maximum cardinality matching. Specifically an *odd set edge cover* is a set of subsets of V with associated edge sets whose union covers all edges in E . The subsets are either singletons i that cover all adjacent edges in $\delta(i)$, or odd subsets S with $|S| \geq 3$ that cover all edges in $E(S)$. The weight of a singleton is 1 and the weight of an odd subset S is $\left\lfloor \frac{|S|}{2} \right\rfloor$.

Corollary 4.1 *The maximum cardinality of a matching equals the minimum weight of an odd set edge cover.*

4.5 Notes

More detailed chapters on assignment problems are in Ahuja et al. [9], and on general matchings in Cook et al. [79] and Nemhauser and Wolsey [233]. For the complete story on both bipartite and nonbipartite matchings, see Lovasz and Plummer [213] and Volume A of Schrijver [267].

- 4.1 The results on alternating paths are from Berge [44] and Norman and Rabin [235].
- 4.3 The primal-dual algorithm can be found in Kuhn [200] and Ford and Fulkerson [125]. Primal dual algorithms for linear programming were also proposed in Dantzig et al. [92]. Many of the first algorithms for combinatorial optimiza-

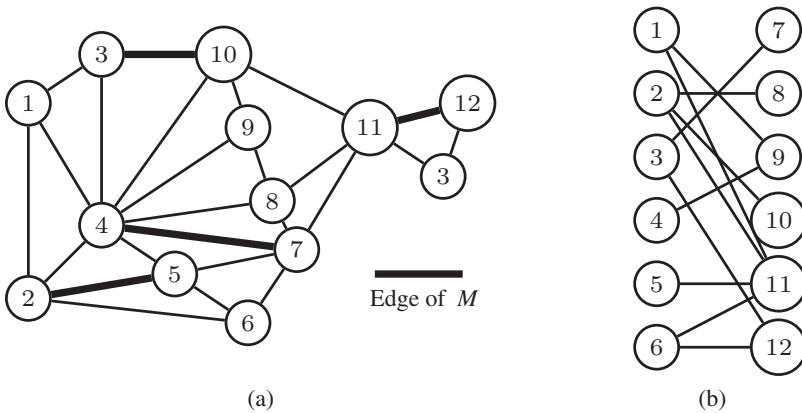


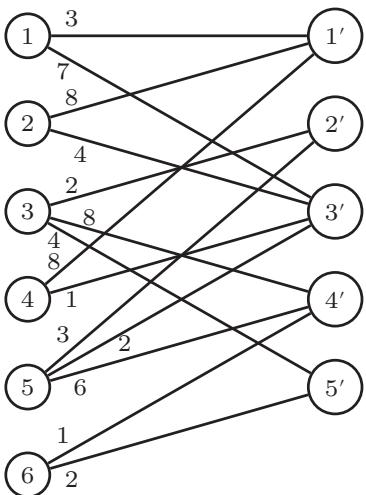
Figure 4.6 (a) Matching to be augmented and (b) maximum cardinality matching.

tion problems were primal-dual, such as the matching algorithm of Edmonds [102] and the matroid intersection algorithm of Edmonds [103].

- 4.4 An algorithm to find a maximum cardinality matching in a general graph is due to Edmonds [101] and an algorithm for the weighted case in [102]. The latter also contains the description of the convex hull of the incidence vectors of the matchings. Padberg and Rao [240] present an efficient algorithm for the separation of the blossom inequalities (4.3).

4.6 Exercises

- Find two augmenting paths for the matching M shown in the graph of Figure 4.6a. Is the new matching M' obtained after augmentation optimal? Why?
- Find a maximum cardinality matching in the bipartite graph of Figure 4.6b. Demonstrate that your solution is optimal.
- 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.
- Show how a maximum flow algorithm can be used to find a maximum cardinality matching in a bipartite graph.

**Figure 4.7** Weighted bipartite graph.

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

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

6. Find a maximum weight matching in the weighted bipartite graph of Figure 4.7.
7. Show how an algorithm for the maximum weight bipartite matching problem can be used to solve the (equality) assignment problem.
8. Find a lower bound on the optimal value of a 6-city TSP instance with distance matrix

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

9. Ten researchers are engaged in a set of 10 projects. Let S_i denote the researchers working on project i for $i = 1, \dots, 10$. To keep track of progress or problems, the 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 $S_1 = \{3, 7, 8, 10\}$, $S_2 = \{4, 8\}$, $S_3 = \{2, 5, 7\}$, $S_4 = \{1, 2, 7, 9\}$, $S_5 = \{2, 5, 7\}$, $S_6 = \{1, 4, 5, 7\}$, $S_7 = \{2, 7\}$, $S_8 = \{1, 6, 7, 10\}$, $S_9 = \{2, 5\}$, $S_{10} = \{1, 2, 3, 6, 7, 8, 10\}$?
10. Suggest a way to solve Exercise 5(i) of Chapter 1.
11. 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.
- 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).
 - 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.
 - 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).
12. Show that the convex hull of perfect matchings in a graph on an even number of nodes is described by the degree constraints (4.1), the nonnegativity constraints (4.2) and the inequalities
- $$\sum_{e \in \delta(S)} x_e \geq 1 \quad \text{for } S \text{ odd, } |S| \geq 3, S \subset V.$$
13.
 - For their annual Christmas party the 30 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 15 rooms?
 - 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?

5

Dynamic Programming

5.1 Some Motivation: Shortest Paths

Here we look at another approach to solving certain combinatorial optimization problems. To see the basic idea, consider the shortest path problem again. Given a directed graph $D = (V, A)$, nonnegative arc distances c_e for $e \in A$, and an initial node $s \in V$, the problem is to find the shortest path from s to every other node $v \in V \setminus \{s\}$. See Figure 5.1, in which a shortest path from s to t is shown, as well as one intermediate node p on the path.

Observation 5.1 If the shortest path from s to t passes by node p , the subpaths (s, p) and (p, t) are shortest paths from s to p , and p to t , respectively.

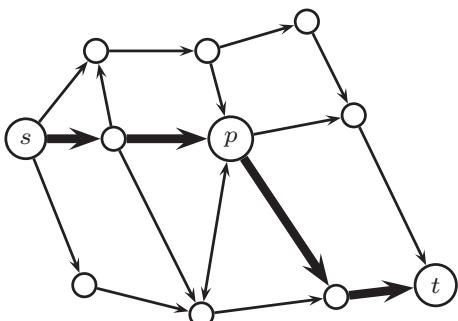
If this were not true, the shorter subpath would allow us to construct a shorter path from s to t , leading to a contradiction.

The question is how to use this idea to find shortest paths.

Observation 5.2 Let $d(v)$ denote the length of a shortest path from s to v . Then

$$d(v) = \min_{i \in V^-(v)} \{d(i) + c_{iv}\}. \quad (5.1)$$

Figure 5.1 Shortest $s - t$ path.



In other words, if we know the lengths of the shortest paths from s to every neighbor (predecessor) of v , then we can find the length of the shortest path from s to v .

This still does not lead to an algorithm for general digraphs because we may need to know $d(i)$ to calculate $d(j)$, and $d(j)$ to calculate $d(i)$. However, for certain digraphs, it does provide a simple algorithm.

Observation 5.3 Given an acyclic digraph $D = (V, A)$ with $n = |V|, m = |A|$, where the nodes are ordered so that $i < j$ for all arcs $(i, j) \in A$, then for the problem of finding shortest paths from node 1 to all other nodes, the recurrence (5.1) for $v = 2, \dots, n$ leads to an $O(m)$ algorithm.

For arbitrary directed graphs with nonnegative weights $c \in R_+^{|A|}$, we need to somehow impose an ordering. One way to do this is to define a more general function $D_k(i)$ as the length of a shortest path from s to i containing at most k arcs. Then we have the recurrence:

$$D_k(j) = \min\{D_{k-1}(j), \min_{i \in V^-(j)} [D_{k-1}(i) + c_{ij}]\}.$$

Now, by increasing k from 1 to $n - 1$, and each time calculating $D_k(j)$ for all $j \in V$ by the recursion, we end up with an $O(mn)$ algorithm and $d(j) = D_{n-1}(j)$.

This approach whereby an optimal solution value for one problem is calculated recursively from the optimal values of slightly different problems is called *Dynamic Programming (DP)*. Below we will see how it is possible to apply similar ideas to derive a recursion for several interesting problems. The standard terminology used is the *Principle of Optimality* for the property that pieces of optimal solutions are themselves optimal, *states* that correspond to the nodes for which values need to be calculated, and *stages* for the steps which define the ordering.

5.2 Uncapacitated Lot-Sizing

The uncapacitated lot-sizing problem (ULS) was introduced in Chapter 1 where two different mixed integer programming formulations were presented. The problem again is to find a minimum cost production plan that satisfies all the nonnegative demands $\{d_t\}_{t=1}^n$, given the costs of production $\{p_t\}_{t=1}^n$, storage $\{h_t\}_{t=1}^n$ and set-up $\{f_t\}_{t=1}^n$. We assume $f_t \geq 0$ for all t .

To obtain an efficient dynamic programming algorithm, it is necessary to understand the structure of the optimal solutions. For this it is useful to view the problem as a network design problem. Repeating the MIP formulation of Section 1.4, we have

$$\min \sum_{t=1}^n p_t y_t + \sum_{t=1}^n h_t s_t + \sum_{t=1}^n f_t x_t \quad (5.2)$$

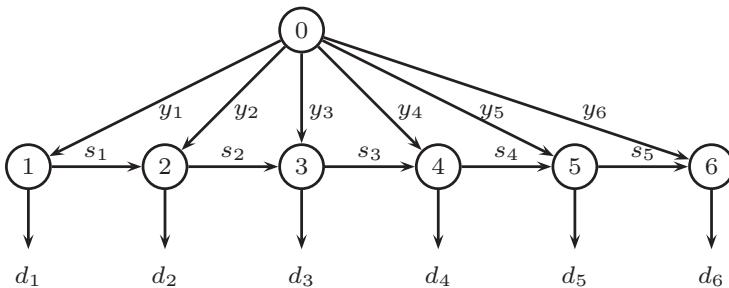


Figure 5.2 Network for lot-sizing problem.

$$s_{t-1} + y_t = d_t + s_t \quad \text{for } t = 1, \dots, n \quad (5.3)$$

$$y_t \leq Mx_t \quad \text{for } t = 1, \dots, n \quad (5.4)$$

$$s_0 = s_n = 0, s \in \mathbb{R}_+^{n+1}, y \in \mathbb{R}_+^n, x \in \{0, 1\}^n, \quad (5.5)$$

where y_t denotes the production in period t and s_t the stock at the end of period t . We see that every feasible solution corresponds to a flow in the network shown in Figure 5.2, where p_t is the flow cost on arc $(0, t)$, h_t is the flow cost on arc $(t, t + 1)$, and the fixed costs f_t are incurred if $y_t > 0$ on arc $(0, t)$.

Thus, ULS is a fixed charge network flow problem in which one must choose which arcs $(0, t)$ are open (which are the production periods) and then find a minimum cost flow through the network. Optimal solutions to ULS have two important structural properties that follow from this network viewpoint.

Proposition 5.1

- (i) *There exists an optimal solution with $s_{t-1}y_t = 0$ for all t . (Production takes place only when the stock is zero.)*
- (ii) *There exists an optimal solution such that if $y_t > 0$, $y_t = \sum_{i=t}^{t+k} d_i$ for some $k \geq 0$. (If production takes place in t , the amount produced exactly satisfies demand for periods t to $t + k$.)*

Proof. Suppose that the production periods have been chosen optimally (certain arcs $(0, t)$ are open). Now, as an optimal extreme flow uses a set of arcs forming a tree, the set of arcs with positive flow contains no cycle, and it follows that only one of the arcs arriving at node t can have a positive flow (i.e. $s_{t-1}y_t = 0$). The second statement then follows immediately. \square

Property (ii) is the important property we need to derive a DP algorithm for ULS.

For the rest of this section, we let d_{it} denote the sum of demands for periods i up to t (i.e., $d_{it} = \sum_{j=i}^t d_j$). The next observation is repeated from Section 1.4 to simplify the calculations.

Observation 5.4 As $s_t = \sum_{i=1}^t y_i - d_{1t}$, the stock variables can be eliminated from the objective function giving $\sum_{t=1}^n p_t y_t + \sum_{t=1}^n h_t s_t = \sum_{t=1}^n p_t y_t + \sum_{t=1}^n h_t (\sum_{i=1}^t y_i - d_{1t}) = \sum_{t=1}^n c_t y_t - \sum_{t=1}^n h_t d_{1t}$, where $c_t = p_t + \sum_{i=t}^n h_i$. This allows us to work with the modified cost function $\sum_{t=1}^n c_t y_t + 0 \sum_{t=1}^n s_t + \sum_{t=1}^n f_t x_t$. Then the constant term $\sum_{t=1}^n h_t d_{1t}$ must be subtracted at the end of the calculations.

Let $H(k)$ be the minimum cost of a solution for periods $1, \dots, k$. If $t \leq k$ is the last period in which production occurs (namely $x_t = d_{tk}$), what happens in periods $1, \dots, t-1$? Clearly, the least cost solution must be optimal for periods $1, \dots, t-1$, and thus has cost $H(t-1)$. This gives the recursion.

Forward Recursion

$$H(k) = \min_{1 \leq t \leq k} \{H(t-1) + f_t + c_t d_{tk}\}$$

with $H(0) = 0$.

Calculating $H(k)$ for $k = 1, \dots, n$ leads to the value $H(n)$ of an optimal solution of ULS. Working back gives a corresponding optimal solution. It is also easy to see that $O(n^2)$ calculations suffice to obtain $H(n)$ and an optimal solution.

Example 5.1 Consider an instance of ULS with $n = 4, d = (2, 4, 5, 1), p = (3, 3, 3, 3), h = (1, 2, 1, 1)$, and $f = (12, 20, 16, 8)$. We start by calculating $c = (8, 7, 5, 4), (d_{11}, d_{12}, d_{13}, d_{14}) = (2, 6, 11, 12)$ and the constant $\sum_{t=1}^4 h_t d_{1t} = 37$.

Now, we successively calculate the values of $H(k)$ using the recursion.

$$H(1) = f_1 + c_1 d_1 = 28.$$

$$H(2) = \min[28 + c_1 d_2, H(1) + f_2 + c_2 d_2] = \min[60, 76] = 60.$$

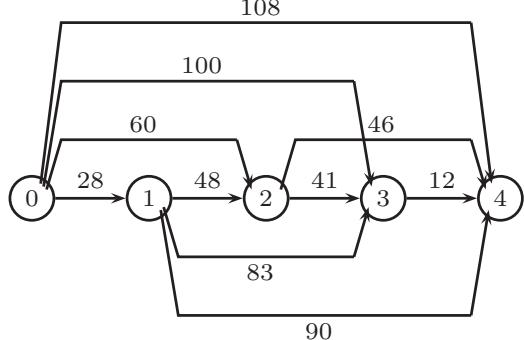
$$H(3) = \min[60 + c_1 d_3, 76 + c_2 d_3, H(2) + f_3 + c_3 d_3] = \min[100, 111, 101] = 100.$$

$$\begin{aligned} H(4) &= \min[100 + c_1 d_4, 111 + c_2 d_4, 101 + c_3 d_4, H(3) + f_4 + c_4 d_4] \\ &= \min[108, 118, 106, 112] = 106. \end{aligned}$$

Working backward, we see that $H(4) = 106 = H(2) + f_3 + c_3 d_{34}$, so $x_3 = 1, y_3 = 6, y_4 = x_4 = 0$. Also, $H(2) = f_1 + c_1 d_{12}$, so $x_1 = 1, y_1 = 6, y_2 = x_2 = 0$. Thus, we have found an optimal solution $y = (6, 0, 6, 0), x = (1, 0, 1, 0), s = (4, 0, 1, 0)$, whose value in the original costs is $106 - 37 = 69$. Checking we have $6p_1 + f_1 + 6p_3 + f_3 + 4h_1 + 1h_3 = 69$. \square

Another possibility is to solve ULS directly as a shortest path problem. Consider a directed graph with nodes $\{0, 1, \dots, n\}$ and arcs (i, j) for all $i < j$. The cost $f_{i+1} +$

Figure 5.3 Shortest path for ULS.



$c_{i+1}d_{i+1,j}$ of arc (i,j) is the cost of starting production in $i + 1$ and satisfying the demand for periods $i + 1$ up to j . Figure 5.3 shows the shortest path instance arising from the data of Example 5.1. Now a least cost path from node 0 to node n provides a minimum cost set of production intervals and solves ULS.

We observe that $H(k)$ is the cost of a cheapest path from nodes 0 to k , and as the directed graph is acyclic, we know from Observation 5.3 that the corresponding shortest path algorithm is $O(m) = O(n^2)$.

5.3 An Optimal Subtree of a Tree

Here we consider another problem that can be tackled by dynamic programming. However, the recursion here is not related at all to shortest paths.

The *Optimal Subtree of a Tree Problem* involves a tree $T = (V, E)$ with a root $r \in V$ and weights c_v for $v \in V$. The problem is to choose a subtree of T rooted at r of maximum weight, or the empty tree if there is no positive weight rooted subtree.

To describe a dynamic programming recursion, we need some notation. For a rooted tree, each node v has a well-defined *predecessor* $p(v)$ on the unique path from the root r to v , and, for $v \neq r$, a set of *immediate successors* $S(v) = \{w \in V : p(w) = v\}$. Also, we let $T(v)$ be the *subtree of T rooted at v* containing all nodes w for which the path from r to w contains v .

For any node v of T , let $H(v)$ denote the optimal solution value of the rooted subtree problem defined on the tree $T(v)$ with node v as the root. If the optimal subtree is empty, clearly $H(v) = 0$. Otherwise, the optimal subtree contains v . It may also contain subtrees of $T(w)$ rooted at w for $w \in S(v)$. By the principle of optimality, these subtrees must themselves be optimal rooted subtrees. Hence, we obtain the recursion:

$$H(v) = \max \left\{ 0, c_v + \sum_{w \in S(v)} H(w) \right\}.$$

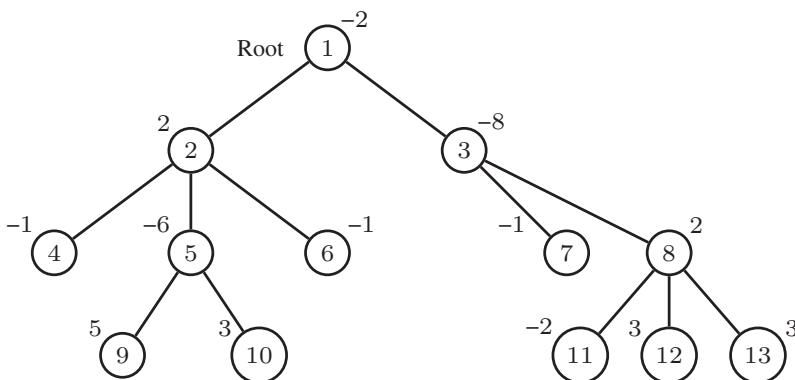


Figure 5.4 Rooted tree with node weights c_v .

To initialize the recursion, we start with the leaves (nodes having no successors) of the tree. For a leaf $v \in V$, $H(v) = \max[c_v, 0]$. The calculations are then carried out by working in from the leaves to the root, until the optimal value $H(r)$ is obtained. As before, an optimal solution is then easily found by working backward out from the root, eliminating every subtree $T(v)$ encountered with $H(v) = 0$. Finally, note that each of the terms c_v and $H(v)$ occurs just once on the right-hand side during the recursive calculations, and so the algorithm is $O(n)$.

Example 5.2 For the instance of the optimal subtree of a tree problem shown in Figure 5.4 with root $r = 1$, we start with the leaf nodes $H(4) = H(6) = H(7) = H(11) = 0$, $H(9) = 5$, $H(10) = 3$, $H(12) = 3$, and $H(13) = 3$. Working in, $H(5) = \max[0, -6 + 5 + 3] = 2$ and $H(8) = \max[0, 2 + 0 + 3 + 3] = 8$. Now, the values of $H(v)$ for all successors of nodes 2 and 3 are known, and so $H(2) = 4$ and $H(3) = 0$ can be calculated. Finally, $H(1) = \max[0, -2 + 4 + 0] = 2$. Cutting off subtrees $T(3)$, $T(4)$, and $T(6)$ leaves an optimal subtree with nodes 1,2,5,9,10 of value $H(1) = 2$. \square

5.4 Knapsack Problems

Here we examine various knapsack problems. Whereas ULS and the optimal subtree problems have the Efficient Optimization Property, knapsack problems in general are more difficult. This is made more precise in the next chapter. Dynamic programming provides an effective approach for such problems if the size of the data is restricted.

5.4.1 0–1 Knapsack Problems

First, we consider the 0–1 knapsack problem:

$$\begin{aligned} Z = \max & \sum_{j=1}^n c_j x_j \\ & \sum_{j=1}^n a_j x_j \leq b \\ & x \in \{0, 1\}^n, \end{aligned}$$

where the coefficients $\{a_j\}_{j=1}^n$ and b are positive integers.

Thinking of the right-hand side λ taking values from $0, 1, \dots, b$ as the state, and the subset of variables x_1, \dots, x_r represented by r as the stage, leads us to define the problem $P_r(\lambda)$ and the optimal value function $f_r(\lambda)$ as follows:

$$\begin{aligned} f_r(\lambda) = \max & \sum_{j=1}^r c_j x_j \\ (P_r(\lambda)) \quad & \sum_{j=1}^r a_j x_j \leq \lambda \\ & x \in \{0, 1\}^r. \end{aligned}$$

Then $Z = f_n(b)$ gives us the optimal value of the knapsack problem. Thus, we need to define a recursion that allows us to calculate $f_r(\lambda)$ in terms of values of $f_s(\mu)$ for $s \leq r$ and $\mu < \lambda$.

What can we say about an optimal solution x^* for problem $P_r(\lambda)$ with value $f_r(\lambda)$? Clearly, either $x_r^* = 0$ or $x_r^* = 1$.

- (i) If $x_r^* = 0$, then by the same optimality argument we used for shortest paths, $f_r(\lambda) = f_{r-1}(\lambda)$.
- (ii) If $x_r^* = 1$, then $f_r(\lambda) = c_r + f_{r-1}(\lambda - a_r)$.

Thus, we arrive at the recursion:

$$f_r(\lambda) = \max\{f_{r-1}(\lambda), c_r + f_{r-1}(\lambda - a_r)\}.$$

Now, starting the recursion with $f_0(\lambda) = 0$ for $\lambda \geq 0$, or alternatively with $f_1(\lambda) = 0$ for $0 \leq \lambda < a_1$ and $f_1(\lambda) = \max[c_1, 0]$ for $\lambda \geq a_1$, we then use the recursion to successively calculate f_2, f_3, \dots, f_n for all integral values of λ from 0 to b .

The question that then remains is how to find an associated optimal solution. For this, we have two related options. In both cases, we iterate back from the optimal value $f_n(b)$. Either we must keep all the $f_r(\lambda)$ values, or an indicator $p_r(\lambda)$ which is 0 if $f_r(\lambda) = f_{r-1}(\lambda)$, and 1 otherwise.

If $p_n(b) = 0$, then as $f_n(b) = f_{n-1}(b)$, we set $x_n^* = 0$ and continue by looking for an optimal solution of value $f_{n-1}(b)$.

Table 5.1 $f_r(\lambda)$ for a 0–1 knapsack problem.

	f_1	f_2	f_3	f_4	p_1	p_2	p_3	p_4
$\lambda = 0$	0	0	0	0	0	0	0	0
1	0	7	7	7	0	1	0	0
2	10	10	10	10	1	0	0	0
3	10	17	17	17	1	1	0	0
4	10	17	17	17	1	1	0	0
5	10	17	17	24	1	1	0	1
6	10	17	25	31	1	1	1	1
7	10	17	32	34	1	1	1	1

If $p_n(b) = 1$, then as $f_n(b) = c_n + f_{n-1}(b - a_n)$, we set $x_n^* = 1$ and then look for an optimal solution of value $f_{n-1}(b - a_n)$.

Iterating n times allows us to obtain an optimal solution.

Counting the number of calculations required to arrive at $Z = f_n(b)$, we see that for each calculation $f_r(\lambda)$ for $\lambda = 0, 1, \dots, b$ and $r = 1, \dots, n$, there are a constant number of additions, subtractions, and comparisons. Calculating the optimal solution requires at most the same amount of work. Thus, the DP algorithm is $O(nb)$.

Example 5.3 Consider the 0–1 knapsack instance:

$$\begin{aligned} Z = \max \quad & 10x_1 + 7x_2 + 25x_3 + 24x_4 \\ & 2x_1 + 1x_2 + 6x_3 + 5x_4 \leq 7 \\ x & \in \{0, 1\}^4. \end{aligned}$$

The values of $f_r(\lambda)$ and $p_r(\lambda)$ are shown in Table 5.1. The values of $f_1(\lambda)$ are calculated by the formula described above. The next column is then calculated from top to bottom using the recursion. For example, $f_2(7) = \max\{f_1(7), 7 + f_1(7 - 1)\} = \max\{10, 7 + 10\} = 17$, and as the second term of the maximization gives the value of $f_2(7)$, we set $p_2(7) = 1$. The optimal value $Z = f_4(7) = 34$.

Working backward, $p_4(7) = 1$ and hence, $x_4^* = 1$. $p_3(7 - 5) = p_3(2) = p_2(2) = 0$ and hence, $x_3^* = x_2^* = 0$. $p_1(2) = 1$ and hence, $x_1^* = 1$. Thus, $x^* = (1, 0, 0, 1)$ is an optimal solution. \square

5.4.2 Integer Knapsack Problems

Now, we consider the integer knapsack problem:

$$\begin{aligned} Z = \max \quad & \sum_{j=1}^n c_j x_j \\ & \sum_{j=1}^n a_j x_j \leq b \\ x & \in \mathbb{Z}_+^n, \end{aligned}$$

where again the coefficients $\{a_j\}_{j=1}^n$ and b are positive integers. Copying from the 0–1 case, we define $P_r(\lambda)$ and the value function $g_r(\lambda)$ as follows:

$$(P_r(\lambda)) \quad \begin{aligned} g_r(\lambda) &= \max \sum_{j=1}^r c_j x_j \\ &\quad \sum_{j=1}^r a_j x_j \leq \lambda \\ &\quad x \in \mathbb{Z}_+^r. \end{aligned}$$

Then $Z = g_n(b)$ gives us the optimal value of the integer knapsack problem.

To build a recursion, a first idea is to again copy from the 0–1 case. If x^* is an optimal solution to $P_r(\lambda)$ giving value $g_r(\lambda)$, then we consider the value of x_r^* . If $x_r^* = t$, then using the principle of optimality $g_r(\lambda) = c_r t + g_{r-1}(\lambda - ta_r)$ for some $t = 0, 1, \dots, \lfloor \frac{\lambda}{a_r} \rfloor$, and we obtain the recursion:

$$g_r(\lambda) = \max_{t=0,1,\dots,\lfloor \lambda/a_r \rfloor} \{c_r t + g_{r-1}(\lambda - ta_r)\}.$$

As $\lfloor \frac{\lambda}{a_r} \rfloor = b$ in the worst-case, this gives an algorithm of complexity $O(nb^2)$.

Can we do better? Is it possible to reduce the calculation of $g_r(\lambda)$ to a comparison of only two cases?

- (i) Taking $x_r^* = 0$, we again have $g_r(\lambda) = g_{r-1}(\lambda)$.
- (ii) Otherwise, we must have $x_r^* \geq 1$. Here we can no longer copy from above.

However, as $x_r^* = 1 + t$ with t a nonnegative integer, we claim, again by the principle of optimality, that if we reduce the value of x_r^* by 1, the resulting vector $(x_1^*, \dots, x_{r-1}^*, t)$ must be optimal for the problem $P_r(\lambda - a_r)$. Thus, we have $g_r(\lambda) = c_r + g_r(\lambda - a_r)$, and we arrive at the recursion:

$$g_r(\lambda) = \max\{g_{r-1}(\lambda), c_r + g_r(\lambda - a_r)\}.$$

This now gives an algorithm of complexity $O(nb)$, which is the same as that of the 0–1 problem. We again set $p_r(\lambda) = 0$ if $g_r(\lambda) = g_{r-1}(\lambda)$ and $p_r(\lambda) = 1$ otherwise.

Example 5.4 Consider the knapsack instance:

$$\begin{aligned} z &= \max 7x_1 + 9x_2 + 2x_3 + 15x_4 \\ 3x_1 + 4x_2 + 1x_3 + 7x_4 &\leq 10 \\ x &\in \mathbb{Z}_+^4. \end{aligned}$$

The values of $g_r(\lambda)$ are shown in Table 5.2. With $c_1 \geq 0$, the values of $g_1(\lambda)$ are easily calculated to be $c_1 \lfloor \frac{\lambda}{a_1} \rfloor$. The next column is then calculated from top to bottom using the recursion. For example, $g_2(8) = \max\{g_1(8), 9 + g_2(8 - 4)\} = \max\{14, 9 + 9\} = 18$.

Table 5.2 $g_r(\lambda)$ for an integer knapsack problem.

	g_1	g_2	g_3	g_4	p_1	p_2	p_3	p_4
$\lambda = 0$	0	0	0	0	0	0	0	0
1	0	0	2	2	0	0	1	0
2	0	0	4	4	0	0	1	0
3	7	7	7	7	1	0	0	0
4	7	9	9	9	1	1	0	0
5	7	9	11	11	1	1	1	0
6	14	14	14	14	1	0	0	0
7	14	16	16	16	1	1	0	0
8	14	18	18	18	1	1	0	0
9	21	21	21	21	1	0	0	0
10	21	23	23	23	1	1	0	0

Working back, we see that $p_4(10) = p_3(10) = 0$ and thus $x_4^* = x_3^* = 0$. $p_2(10) = 1$ and $p_2(6) = 0$ and so $x_2^* = 1$. $p_1(6) = p_1(3) = 1$ and thus $x_1^* = 2$. Hence, we obtain an optimal solution $x^* = (2, 1, 0, 0)$. \square

Another recursion can be used for the integer knapsack problem. Looking at the example above, we see that in fact all the important information is contained in the n th column containing the values of $g_n(\lambda)$. Writing h in place of g_n , can we directly write a recursion for $h(\lambda)$?

Again, the principle of optimality tells us that if x^* is an optimal solution of $P_n(\lambda)$ of value

$$h(\lambda) = \max \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq \lambda, x \in \mathbb{Z}_+^n \right\}$$

with $x_j^* \geq 1$, then $h(\lambda) = c_j + h(\lambda - a_j)$.

Thus, we obtain the recursion:

$$h(\lambda) = \max_{j: a_j \leq \lambda} [h(\lambda - a_j) + c_j] \quad \text{for } \lambda \in [1, b] \quad (5.6)$$

with $h(0) = 0$. This also leads to an $O(nb)$ algorithm. Applied to the instance of Example 5.4, it gives precisely the values in the g_4 column of Table 5.2.

A Longest Path Extended Formulation

The dynamic programming approach for knapsack problems can also be viewed as a longest path problem. For the integer knapsack problem with the recursion (5.6), construct an acyclic digraph $D = (V, A)$ with nodes $0, 1, \dots, b$, arcs $(\lambda, \lambda + a_j)$

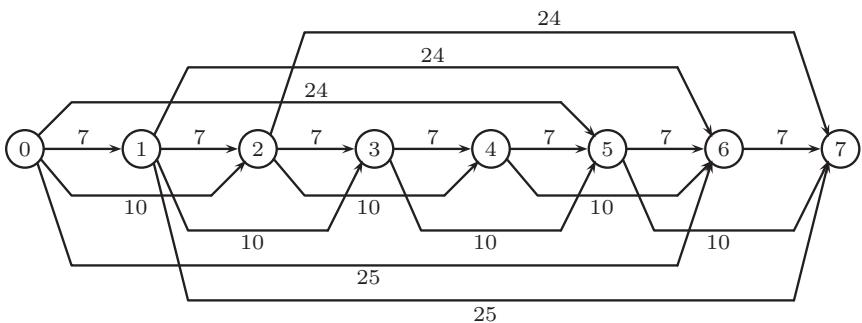


Figure 5.5 Knapsack longest path problem.

for $\lambda \in \mathbb{Z}_+^1$, $\lambda \leq b - a_j$ with weight c_j for $j = 1, \dots, n$, and 0-weight arcs $(\lambda, \lambda + 1)$ for $\lambda \in \mathbb{Z}_+^1$, $\lambda \leq b - 1$. $h(\lambda)$ is precisely the value of a longest path from node 0 to node λ . Figure 5.5 shows the digraph arising from the instance:

$$\begin{aligned} Z &= \max 10x_1 + 7x_2 + 25x_3 + 24x_4 \\ 2x_1 + 1x_2 + 6x_3 + 5x_4 &\leq 7 \\ x &\in \mathbb{Z}_+^4, \end{aligned}$$

except that the 0-weight arcs $(\lambda, \lambda + 1)$ are omitted by dominance.

To obtain an extended formulation, let $z(j, \lambda) = 1$ if the path includes the arc $(\lambda, \lambda + a_j)$ and $z(0, \lambda) = 1$ if it uses the arc $(\lambda, \lambda + 1)$. This gives

$$\begin{aligned} Z &= \max \sum_{j=1}^n c_j x_j \\ \sum_{0 \leq j \leq n: \lambda \geq a_j} z(j, \lambda - a_j) - \sum_{0 \leq j \leq n: \lambda + a_j \leq b} z(j, \lambda) &= 0 \quad \text{for } \lambda = 1, \dots, b - 1 \\ \sum_{j=0}^n z(j, 0) &= 1 \\ x_j - \sum_{\lambda} z(j, \lambda) &= 0 \quad \text{for } j = 1, \dots, n \\ x \in \mathbb{R}^n, z \in \{0, 1\}^{(n+1)b}. \end{aligned}$$

As the matrix corresponding to the z -variables is TU, the linear programming relaxation in which the integrality of the z -variables is dropped has integral solutions and thus solves the knapsack problem.

5.5 The Cutting Stock Problem*

This well-known problem can be stated as follows: Given an unlimited number of sheets of length L and demands for q_j pieces of length a_j for $j = 1, \dots, n$, propose a cutting plan that uses the minimum number of sheets of length L . We present briefly two formulations.

Formulation 1 Let $\{x^t\}_{t=1}^T$ be the set of all cutting patterns, where x_j^t is the number of pieces of length a_j in pattern t . In other words, these are all the feasible solutions to the knapsack set $X = \{x \in \mathbb{Z}_+^n : \sum_{j=1}^n a_j x_j \leq L\}$. The problem can now be formulated as

$$\begin{aligned} Z^{\text{CS}} &= \min \sum_{t=1}^T w_t \\ &\sum_{t=1}^T x^t w_t \geq q \\ w &\in \mathbb{Z}_+^T, \end{aligned}$$

where w_t is the number of times that pattern x^t is selected. Note that this formulation involves an exponential number of variables. Ways to treat such formulations are discussed in Chapter 11.

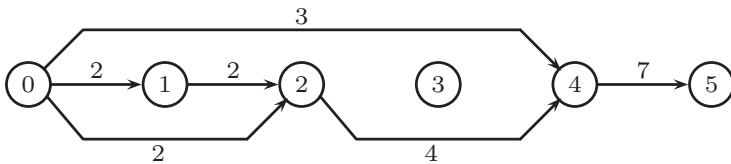
Formulation 2 Here we make use of the extended formulation for the knapsack problem from the previous section. This uses the nontrivial property that an integral $s - t$ network flow of size K can be decomposed into K unit $s - t$ flows. The problem is then to find the smallest possible flow between nodes 0 and L such that each type j arc representing a piece of length a_j appears at least q_j times. This leads to the formulation:

$$\begin{aligned} Z^{\text{CS}} &= \min \eta \\ \sum_{j=0}^n z(j, 0) - \eta &= 0 \\ \sum_{0 \leq j \leq n: \lambda \geq a_j} z(j, \lambda) - \sum_{0 \leq j \leq n: \lambda + a_j \leq b} z(j, \lambda) &= 0 \quad \text{for } \lambda = 1, \dots, b-1 \\ \sum_{\lambda} z(j, \lambda) &\geq q_j \quad \text{for } j = 1, \dots, n \\ \eta \in \mathbb{Z}_+^1, z \in \mathbb{Z}_+^{(n+1)b}. \end{aligned}$$

Example 5.5 Consider an instance with standard pieces of length $L = 5$. There is demand for $n = 3$ different lengths $a = (1, 2, 4)$ and the number required is $q = (10, 6, 3)$. Noting that it suffices to take just maximal cutting patterns, the first formulation gives

$$\begin{aligned} \min w_1 + w_2 + w_3 + w_4 \\ \begin{pmatrix} 5 \\ 0 \\ 0 \end{pmatrix} w_1 + \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} w_2 + \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix} w_3 + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} w_4 \geq \begin{pmatrix} 10 \\ 6 \\ 3 \end{pmatrix} \\ w \in \mathbb{Z}_+^4 \end{aligned}$$

with optimal solution $w = (0, 2, 2, 3)$ of value 7. In Figure 5.6, a feasible flow of 7 units is shown for formulation 2 as well as its decomposition giving the cutting patterns. \square



$$z(2, 0) = z(2, 2) = z(1, 4) = 2: \quad x^2 = (1, 2, 0), w_2 = 2$$

$$z(1, 0) = z(1, 1) = z(2, 2) = z(1, 4) = 2: \quad x^3 = (3, 1, 0), w_3 = 2$$

$$z(3, 0) = z(1, 4) = 3: \quad x^4 = (1, 0, 1), w_4 = 3$$

Figure 5.6 Decomposition of integer flow of 7 units.

5.6 Notes

- 5.1 The principle of optimality and dynamic programming originated with Bellman [39]. One of the more recent books on dynamic programming is Bertsekas [48]. Shortest path problems with additional restrictions that arise as subproblems in many routing problems are solved by dynamic programming, see Desrosiers et al. [97].
- 5.2 The uncapacitated lot-sizing model and the dynamic programming algorithm for it are from Wagner–Whitin [288]. Dynamic programming recursions have been developed for many generalizations including lot-sizing with backlogging by Zangwill [303], with constant production capacities by Florian and Klein [123], with start-up costs by Fleischmann [122] and with lower bounds on production by Constantino [77]. Wagelmans et al. [287] among others have shown how the running time of the DP algorithm for the basic model can be significantly improved to $O(n \log n)$ and even $O(n)$ in certain cases.
- 5.3 Various generalizations of the subtree problem are of interest. The more general problem of finding an optimal upper/lower set in a partially ordered set is examined in Gröflin and Liebling [161] and shown to be solvable as a network flow problem.
The problem of partitioning a tree into subtrees is tackled using dynamic programming in Barany et al. [33] and this model is further studied in Aghezzaf et al. [8]. A telecommunications problem with such structure is studied in Balakrishnan et al. [19].
Many other combinatorial optimization problems become easy when the underlying graph is a tree, see Magnanti and Wolsey [217], or more generally when it is a series-parallel graph, see Takamizawa et al. [273].
- 5.4 The classical paper on the solution of knapsack problems by dynamic programming is by Gilmore and Gomory [143]. The longest path or dynamic

programming viewpoint was later extended by Gomory leading to the group relaxation of an integer program [156], and later to the superadditive duality theory for integer programs; see Notes of Section 2.5. The idea of reversing the roles of the objective and constraint rows (Exercise 7) is from Ibarra and Kim [181].

- 5.5 The first cutting stock formulation is due to Gilmore and Gomory [141] and the second appeared in Wolsey [296], see also Valerio de Carvalho [277]. Computational results with the latter pseudo-polynomial formulation and graph compression can be found in Branda and Pedroso [60].

The dynamic programming approach to TSP (Exercise 9) was first described by Held and Karp [173]. Relaxations of this approach, known as *state space relaxation*, have been used for a variety of constrained path and routing problems; see Christofides et al. [68]. Psaraftis [252] and Balas [24] show how such a recursion is of practical interest when certain restrictions on the tours or arrival sequences are imposed.

5.7 Exercises

- Solve the uncapacitated lot-sizing problem with $n = 4$ periods, unit production costs $p = (1, 1, 1, 2)$, unit storage costs $h = (1, 1, 1, 1)$, set-up costs $f = (20, 10, 45, 15)$, and demands $d = (8, 5, 13, 4)$.
- Consider the uncapacitated lot-sizing problem with backlogging (ULSB). *Backlogging* means that demand in a given period can be satisfied from production in a later period. If $r_t \geq 0$ denotes the amount backlogged in period t , the flow conservation constraints (5.3) become

$$s_{t-1} - r_{t-1} + y_t = d_t + s_t - r_t.$$

Show that there always exists an optimal solution to ULSB with

- (i) $s_{t-1}y_t = y_t r_t = s_{t-1}r_t = 0$.
- (ii) $y_t > 0$ implies $y_t = \sum_{i=p}^q d_i$ with $p \leq t \leq q$.

Use this to derive a dynamic programming recursion for ULSB, or to reformulate ULSB as a shortest path problem.

- Find a maximum weight rooted subtree for the rooted tree shown in Figure 5.7.
- Formulate the optimal subtree of a tree problem as an integer program. Is this IP easy to solve?

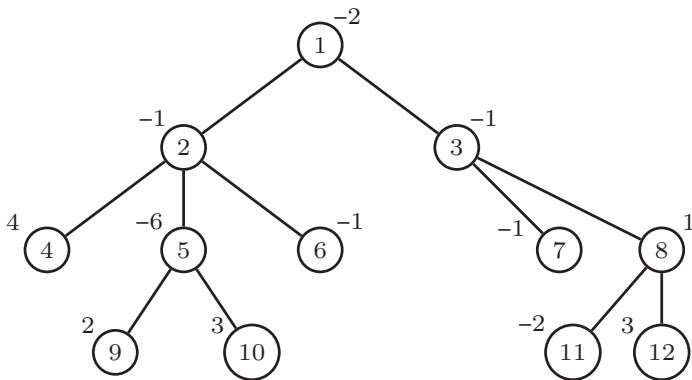


Figure 5.7 Rooted tree with node weights c_v .

5. Given a digraph $D = (V, A)$, travel times c_{ij} for $(i, j) \in A$ for traversing the arcs, and earliest passage times r_j for $j \in V$, consider the problem of minimizing the time required to go from node 1 to node n .
 - (i) Describe a dynamic programming recursion.
 - (ii) Formulate as a mixed integer program. Is this mixed integer program easy to solve?
6. Solve the knapsack problem

$$\begin{aligned} & \min 5x_1 - 3x_2 + 7x_3 + 10x_4 \\ & 2x_1 - x_2 + 4x_3 + 5x_4 \geq \lambda \\ & x_1, x_4 \in \mathbb{Z}_+^1, x_2, x_3 \in \{0, 1\} \end{aligned}$$

for all values of λ between 7 and 10.

7. Let $f(\lambda) = \max\{\sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq \lambda, x \in X\}$ with $X \subseteq \mathbb{Z}_+^n$ and $h(t) = \min\{\sum_{j=1}^n a_j x_j : \sum_{j=1}^n c_j x_j \geq t, x \in X\}$. Show that
 - (i) $f(\lambda) \geq t$ if and only if $h(t) \leq \lambda$.
 - (ii) $f(b) = \max\{t : h(t) \leq b\}$.

Use these observations to solve the knapsack problem

$$\begin{aligned} & \max 3x_1 + 4x_2 + 6x_3 + 5x_4 + 8x_5 \\ & 412x_1 + 507x_2 + 714x_3 + 671x_4 + 920x_5 \leq 1794 \\ & x \in \{0, 1\}^5 \end{aligned}$$

by dynamic programming.

- 8.** Solve the problem

$$\max x_1^3 + 2x_2^{\frac{1}{2}} + 4x_3 + 4x_4$$

$$2x_1^2 + 4x_2 + 6x_3 + 5x_4 \leq t$$

$$x_1 \leq 3, x_2 \leq 2, x_4 \leq 1$$

$$x \in \mathbb{Z}_+^4$$

for $t \leq 12$. (Hint. One of the recursions in the chapter is easily adapted.)

- 9.** 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 .
- 10.** Given the weighted rooted subtree of Figure 5.7, devise a dynamic programming algorithm to find optimal weighted subtrees with $k = 2, 3, \dots, n - 1$ nodes.
- 11.** 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.
- 12.** * 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 (Chapter 4)
- 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 Cutting Stock Problem (Chapter 5)

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 11 in Chapter 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. Then we will 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, in this and Section 6.3, when we refer to an optimization problem, we have in mind the corresponding decision problem. We will use the notation *TSP*, *UFL*

and IP , etc. to denote the class of decision problems associated to the optimization problems TSP, UFL and IP respectively.

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, posed as a maximization, the associated decision problem is “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*. A dynamic programming algorithm for ULS 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, $ULS \in \mathcal{P}$.

- (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 0–1 KNAPSACK $\in \mathcal{NP}$.

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 an instance of STSP with data $(G, c^{|E|}, k)$ with minimization, it suffices to check that a proposed set of edges forms a tour and that its length does not exceed k . So STSP $\in \mathcal{NP}$. 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 appropriate solution x^* whose description length $\sum_{j=1}^n \lceil \log x_j^* \rceil$ is polynomial in L . \square

Do the optimization problems in the second set 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{NP} is nonempty, but that all of the decision problems in our second list are in \mathcal{NP} . So how can one prove that a problem is in \mathcal{NP} ?

The most important step is to prove that \mathcal{NP} is nonempty. Written as an 0–1 integer program, an instance of *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:

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

$$x \in \{0, 1\}^n$$

have a feasible solution?

It is obvious that *SATISFIABILITY* $\in \mathcal{NP}$. Cook showed in 1970 that *SATISFIABILITY* $\in \mathcal{NP}$.

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{NP} .

For example, to see that 0-1 IP is in \mathcal{NP} , all we need to observe is that

- (i) 0-1 IP $\in \mathcal{NP}$, (which is immediate) and
- (ii) *SATISFIABILITY* reduces to 0-1 IP. (Above we actually described *SATISFIABILITY* as a 0-1 integer program, 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 $P \in \mathcal{NP}$ and P is polynomially reducible to Q , then $Q \in \mathcal{NP}$.

Proof. (ii) Consider any problem $R \in \mathcal{NP}$. As $P \in \mathcal{NP}$, 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 $R \in \mathcal{NP}$, $Q \in \mathcal{NP}$. \square

This has an important corollary.

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

Proof. Suppose $Q \in \mathcal{P} \cap \mathcal{NP}$ and take $R \in \mathcal{NP}$. As $R \in \mathcal{NP}$ and $Q \in \mathcal{NP}$, 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 $\mathcal{P} = \mathcal{NP}$. \square

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 show that the *CLS* and *NODE COVER* can be added to the list.

Example 6.2 We show that *CLS* $\in \mathcal{NPC}$.

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

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

First, is *CLS* $\in \mathcal{NP}$? One answer lies in the observation that there is always an optimal solution of the form: $x \in \{0, 1\}^n$ with y a basic feasible solution of the network flow problem in which x 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:

$$\min \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \geq b, x \in \{0, 1\}^n \right\},$$

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

$$\begin{aligned} \min & \sum_{t=1}^n p'_t y_t + \sum_{t=1}^n f_t x_t \\ \sum_{i=1}^t y_i &\geq \sum_{i=1}^t d_i \quad \text{for } t = 1, \dots, n-1 \\ \sum_{i=1}^n y_i &= \sum_{i=1}^n d_i \\ y_t &\leq C_t x_t \quad \text{for } t = 1, \dots, n \\ y \in \mathbb{R}_+^n, x &\in \{0, 1\}^n. \end{aligned}$$

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

$$\begin{aligned} \min & \sum_{t=1}^n c_t x_t \\ \sum_{i=1}^t y_i & \geq 0 \quad \text{for } t = 1, \dots, n-1 \\ \sum_{i=1}^n y_i & = b \\ y_t & \leq a_t x_t \quad \text{for } t = 1, \dots, n \\ y & \in \mathbb{R}_+^n, x \in \{0, 1\}^n. \end{aligned}$$

Dropping the $n - 1$ redundant demand constraints leaves

$$\begin{aligned} \min & \sum_{t=1}^n c_t x_t \\ \sum_{t=1}^n y_t & = b \\ y_t & \leq a_t x_t \quad \text{for } t = 1, \dots, n \\ y & \in \mathbb{R}_+^n, x \in \{0, 1\}^n. \end{aligned}$$

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

Example 6.3 $NODE\ COVER \in \mathcal{NPC}$, where the associated decision problem is:

Given $G = (V, E)$, does there exist $U \subseteq V$ such that every edge is incident to a node of U and $|U| \leq k$?

First, we consider the problem $3-SAT$ which is the special case of $SATISFIABILITY$ that we rewrite in the form

$$\sum_{j \in C_i} x_j + \sum_{j \in D_i} \bar{x}_j \geq 1 \quad \text{for } i = 1, \dots, m,$$

where $\bar{x}_j = 1 - x_j$ and in which $|C_i \cup D_i| = 3$ with C_i, D_i disjoint for all i . It is easy to see how to reduce $SATISFIABILITY$ to $3-SAT$ in polynomial time and so $3-SAT \in \mathcal{NPC}$.

Now, we reduce $3-SAT$ to $NODE\ COVER$. Given an instance of $3-SAT$, we associate the following instance of $NODE\ COVER$. First, we describe the nodes. For

each clause i , there are three nodes v_{i1}, v_{i2}, v_{i3} . Also create $2n$ nodes representing each of the terms x_j and \bar{x}_j for $j = 1, \dots, n$. The edges are as follows: $f_{ikk'} = (v_{ik}, v_{ik'})$ for $1 \leq k < k' \leq 3$. Also, there is an edge $h_j = (x_j, \bar{x}_j)$ for $j = 1, \dots, n$. If the k th term of clause i is the term x_j , there is an edge $g_{ik} = (v_{ik}, x_j)$ and if it is the term \bar{x}_j , there is an edge $g_{ik} = (v_{ik}, \bar{x}_j)$.

CLAIM. The 3-SAT instance is satisfiable if and only if there is a node cover of size $n + 2m$.

Note first that either two or three of the nodes v_{ik} are needed to cover the edges $f_{ikk'}$ for each i and at least one of the nodes x_j, \bar{x}_j is needed to cover edge h_j .

Suppose that the 3-SAT instance is satisfiable with assignment $x_j = 1$ for $j \in S$ and $x_j = 0$ otherwise. Select nodes x_j for $j \in S$ and nodes \bar{x}_j for $j \in \{1, \dots, n\} \setminus S$. As clause i is satisfied, one of the edges g_{ik} (at least) is covered. It suffices to select the two nodes $v_{ik'}$ and $v_{ik''}$ with $\{k', k''\} = \{1, 2, 3\} \setminus \{k\}$ to obtain a node cover with $n + 2m$ nodes.

Conversely, consider a node cover with $n + 2m$ nodes. As at least n nodes are needed to cover the edges of type h_j for $j = 1, \dots, n$ and at least $2m$ to cover the edges of type $f_{ikk'}$, it follows that exactly one node, x_j or \bar{x}_j covers the h_j edge, and two v_{ik} nodes cover the remaining edges associated to clause i . But then the assignment provided by the nodes covering h_j for $j = 1, \dots, n$ satisfies each of the clauses $i = 1, \dots, m$ and the 3-SAT instance is satisfied.

The corresponding node cover problem written as an IP is

$$\begin{aligned} \min \sum_j (x_j + \bar{x}_j) + \sum_{i,k} v_{ik} \\ x_j + \bar{x}_j \geq 1 \text{ for } j = 1, \dots, n \\ x_j + v_{ik} \geq 1 \text{ if term } k \text{ of clause } i \text{ is } x_j \text{ for } i = 1, \dots, m, k = 1, 2, 3 \\ \bar{x}_j + v_{ik} \geq 1 \text{ if term } k \text{ of clause } i \text{ is } \bar{x}_j \text{ for } i = 1, \dots, m, k = 1, 2, 3 \\ v_{ik} + v_{ik'} \geq 1 \text{ for } i = 1, \dots, m, 1 \leq k < k' \leq 3 \\ x, \bar{x} \in \{0, 1\}^n, v \in \{0, 1\}^{3m}. \end{aligned}$$

The 3-SAT instance with $n = 3$ and $m = 4$

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

is satisfiable with $\bar{x}_1 = x_2 = x_3 = 1$. The corresponding graph is shown in Figure 6.1, where starred nodes show a corresponding node cover with $n + 2m = 11$ nodes. \square

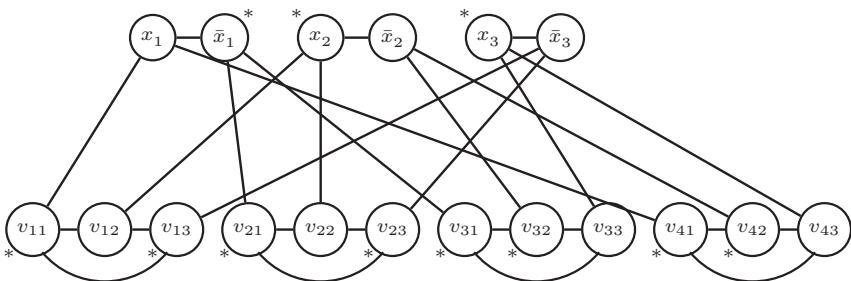


Figure 6.1 Node Cover for 3-SAT.

6.4 Consequences of $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$

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

So how should we interpret the above results and observations?

A first important remark concerns the class \mathcal{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 \mathcal{NP}), 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 (courageous)* might set out to become famous by proving that $\mathcal{P} \neq \mathcal{NP}$.

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 John 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 20 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, discuss briefly the existence of polynomial size extended formulations and close with a few examples of approximation algorithms (analysis of polynomial time heuristics with worst-case performance guarantees).

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 $S \subseteq \{0, 1\}^n$), 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 The Complexity of Extended Formulations

In earlier chapters we have seen that, for several optimization problems $\max\{cx : x \in X\}$, there exist extended formulations of polynomial or pseudo-polynomial size describing $\text{conv}(X)$: among others the uncapacitated lot-sizing problem in Section 1.7 and the maximum weight spanning tree problem in Section 3.5 of polynomial size and the integer knapsack problem in Section 5.4 of pseudo-polynomial size. In particular, if $Q = \{(x, w) \in \mathbb{R}^n \times \mathbb{R}^p : Ax + Gw \leq d\}$ is such an extended formulation and $\text{proj}_x(Q) = \text{conv}(X) \subseteq \mathbb{R}^n$, the following useful results follow:

1. The optimization problem $\max\{cx : x \in X\}$ can be solved by the linear program: $\max\{cx + 0w : Ax + Gw \leq d\}$.
2. The separation problem: Is $x^* \in \text{conv}(X)$ can be solved by solving the LP: $\max\{0w : Gw \leq d - Ax^*\}$. Either it is feasible and $x^* \in \text{conv}(X)$, or it is infeasible and the linear program provides an extreme ray v of the set $\{u \in \mathbb{R}_+^m : uG = 0\}$ with $v(d - Ax^*) < 0$. The valid inequality $v(d - Ax) \geq 0$ then cuts off the point x^* .

For the problems in which the extended formulation Q is of polynomial size, the (associated decision) problem is necessarily in \mathcal{P} . This leads naturally to the question whether there is a polynomial size extended formulation for $\text{conv}(X)$ for all optimization problems in \mathcal{P} .

We now present two recent very impressive, but negative results (whose proofs are beyond the scope of this book) related to this question.

Theorem 6.1 *Every extended formulation for the perfect matching polytope requires at least $O(2^{cn})$ constraints for some $c > 0$.*

This has as a corollary that the same holds for the STSP polytope.

Theorem 6.2 *Every extended formulation for the stable set polytope has at least $(1.5)^{\sqrt{n}}$ constraints.*

So Theorem 6.1 tells us that for problems in \mathcal{P} , there is no guarantee of a compact extended formulation. The corollary and Theorem 6.2 tell us that we are wasting

our time if we are looking for a polynomial-size linear program to solve matching, STSP or stable set problems.

6.7 Worst-Case Analysis of Heuristics*

Here we take the viewpoint of the thoughtful mathematician: The goal is to find an *approximation algorithm* that runs in polynomial time and guarantees a solution of a certain quality.

To start with, we consider a very simple example, the integer knapsack problem:

$$(IKP) \quad Z = \max \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x \in \mathbb{Z}_+^n \right\},$$

where $\{a_j\}_{j=1}^n, b \in \mathbb{Z}_+$. Without loss of generality, we assume that $a_j \leq b$ for $j \in N$ and $\frac{c_1}{a_1} \geq \frac{c_j}{a_j}$ 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 $Z^H = cx^H$.

Theorem 6.3 $\frac{Z^H}{Z} \geq \frac{1}{2}$.

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 $Z^{LP} = \frac{c_1 b}{a_1} \geq Z$.

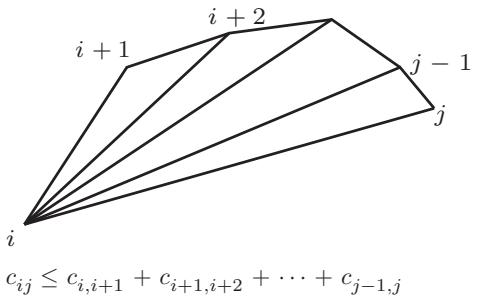
Now, as $a_1 \leq b$, $\lfloor \frac{b}{a_1} \rfloor \geq 1$. Setting $\frac{b}{a_1} = \lfloor \frac{b}{a_1} \rfloor + f$ with $0 \leq f < 1$, we have that $\lfloor \frac{b}{a_1} \rfloor / \frac{b}{a_1} = \frac{\lfloor \frac{b}{a_1} \rfloor}{\lfloor \frac{b}{a_1} \rfloor + f} \geq \frac{\lfloor \frac{b}{a_1} \rfloor}{\lfloor \frac{b}{a_1} \rfloor + \lfloor \frac{b}{a_1} \rfloor} = \frac{1}{2}$. So $\frac{Z^H}{Z} \geq \frac{Z^H}{Z^{LP}} = \frac{c_1 \lfloor \frac{b}{a_1} \rfloor}{c_1 \frac{b}{a_1}} = \frac{\lfloor \frac{b}{a_1} \rfloor}{\frac{b}{a_1}} \geq \frac{1}{2}$. \square

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 6.2).

To understand the heuristics and their analysis, we need to introduce Eulerian graphs.

Definition 6.8 $G = (V, E)$ is a *Eulerian graph* if the degree of each node is even.

Figure 6.2 Triangle Inequality.

Proposition 6.4 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 6.5 Given a complete graph H on node set V with edge lengths satisfying the triangle inequality, let $G = (V, E)$ be a connected Eulerian subgraph 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 $v_{i_1}, v_{i_2}, \dots, v_{i_n}, v_{i_{n+1}}$ is a tour with $v_{i_1} = v_{i_{n+1}}$. 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 $\sum_{k=1}^n c_{f_k} \leq \sum_{k=1}^n \sum_{j=i_k+1}^{i_{k+1}} c_{e_j} = \sum_{e \in E} c_e$. \square

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 6.5, 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 6.6 $\frac{Z^H}{Z} \leq 2$.

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 $\frac{Z^H}{2} \leq Z_T \leq Z \leq Z^H$. \square

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 6.5, convert the Eulerian graph into a tour of length Z^C .

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.

Proposition 6.7 $\frac{Z^C}{Z} \leq \frac{3}{2}$.

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 matchings $M_1 = \{(j_1, j_2), (j_3, j_4), \dots, (j_{2k-1}, j_{2k})\}$ of length

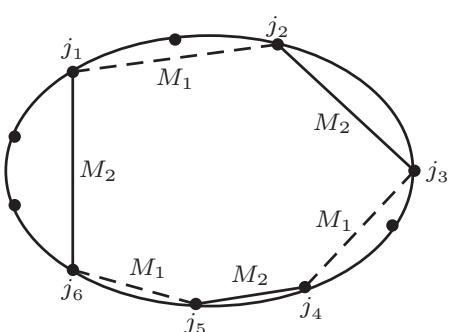


Figure 6.3 Optimal tour longer than two matchings on V' .

Z_{M_1} and $M_2 = \{(j_2, j_3), (j_4, j_5), \dots, (j_{2k}, j_1)\}$ 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 6.3). 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 6.5, $Z^C \leq Z_T + Z_M \leq Z + Z/2 = (3/2)Z$. \square

6.8 Notes

- 6.2 An important step in understanding the distinction between easy and difficult problems is the concept of a certificate of optimality, see Edmonds [101, 102].
- 6.3 Cook [78] 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 Karp [193, 194]. The book of Garey and Johnson [134] lists an enormous number of \mathcal{NP} -complete problems and their reductions, including that from 3-SAT to NODE COVER. For a site with a 2005 update, see Crescenzi and Kann [85].
- 6.5 The equivalence of optimization and separation is shown in Grötschel et al. [162, 163]. A more thorough exploration of the equivalence appears in their book [164]. Other results of importance for integer programming concern the difficulty of finding a short description of all facets for \mathcal{NP} -hard problems Papadimitriou and Yannakakis [244], and the polynomiality of integer programming with a fixed number of variables Lenstra [206]. Some separation problems are examined in Chapter 9.
- 6.6 Yannakakis [302] showed that there is no polynomial-size extended formulation for the matching polytope under certain symmetry assumptions. He showed that the fundamental importance of two concepts and a result that is the basis of later work: the *nonnegative rank* of the *slack matrix* associated with a polytope P gives a lower bound on the number of constraints needed in an extended formulation of P . Theorem 6.1 is due to Rothvoss [260] and Theorem 6.2 to Fiorini et al. [110]. Following on these important results it has been shown that several other \mathcal{NP} -hard problems cannot have polynomial size extended formulations.
- 6.7 The first worst-case analysis of a heuristic for a scheduling problem is in Graham [160], but the analysis of bin packing heuristics of Johnson et al. [187] seems to have initiated much of the work in this area. The tree/matching heuristic for STSP is from Christofides [67].

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 $1/\alpha$.

For two recent books on the design and analysis of approximation algorithms, see Vazariani [286] and Williamson and Shmoys [293].

6.9 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 $\sum_{j \in S} a_j = \sum_{j \in N \setminus S} a_j$, or prove that it is impossible. Show that 2-PARTITION is polynomially reducible to 0–1 KNAPSACK. Does this imply that 2-PARTITION is \mathcal{NP} -complete?
2. Show that SATISFIABILITY is polynomially reducible to STABLE SET and thus that STABLE SET is \mathcal{NP} -complete, where STABLE SET is the decision problem related to finding a maximum weight set of nonadjacent nodes in a graph.
3. Show that STABLE SET is polynomially reducible to SET PACKING, where SET PACKING is the decision problem related to finding a maximum weight set of disjoint columns in a 0–1 matrix.
4. Show that SATISFIABILITY is polynomially reducible to 3-SAT.
5. Show that SET COVERING is polynomially reducible to UFL.
6. Show that SET COVERING is polynomially reducible to DIRECTED STEINER TREE.
7. Given $D = (V, A)$, c_e for $e \in A$, a subset $F \subseteq A$, and a node $r \in V$, ARC ROUTING is the decision problem related to finding a minimum length directed subtour that contains the arcs in F and starts and ends at node r . Show that TSP is polynomially reducible to ARC ROUTING.
8. * Show that the decision problem associated to IP is an integer programming feasibility problem and is in \mathcal{NP} .
9. Consider a 0–1 knapsack set $X = \{x \in \{0, 1\}^n : \sum_{j \in N} a_j x_j \leq b\}$ with $0 \leq a_j \leq b$ for $j \in N$ and let $\{x^t\}_{t=1}^T$ be the points of X . With it, associate the bounded polyhedron $\Pi^1 = \{\pi \in \mathbb{R}_+^n : x^t \pi \leq 1 \text{ for } t = 1, \dots, T\}$ with extreme points $\{\pi^s\}_{s=1}^S$. Consider a point x^* with $0 \leq x_j^* \leq 1$ for $j \in N$.)

- (i) Show that $x^* \in \text{conv}(X)$ if and only if $\min\{\sum_{t=1}^T \lambda_t : x^* \leq \sum_{t=1}^T x^t \lambda_t, \lambda \in \mathbb{R}_+^T\} = \max\{x^* \pi : \pi \in \Pi^1\} \leq 1$.
- (ii) Deduce that if $x^* \notin \text{conv}(X)$, then for some $s = 1, \dots, S$, $\pi^s x^* > 1$.
10. Compare with Definition 6.3. $\text{co-}\mathcal{NP}$ is the class of decision problems with the property that: for any instance for which the answer is NO, there is a “short” (polynomial) proof of the NO. Verify that several of the decision problems associated to the problems with the Efficient Optimization property in the first list at the beginning of the chapter lie in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$.
11. Consider the 0–1 knapsack problem: $Z = \max\{\sum_{j \in N} c_j x_j : \sum_{j \in N} a_j x_j \leq b, x \in \{0, 1\}^n\}$ with $0 < a_j \leq b$ for $j \in N$. Consider a greedy heuristic that chooses the better of the integer round down of the linear programming solution, and the best solution in which just one variable $x_k = 1$, where $c_k = \max_j c_j$. Show that $Z^G \geq \frac{1}{2}Z$.
12. Consider the problem of finding a maximum cardinality matching from Chapter 4. A matching $M \subseteq E$ is *maximal* if $M \cup \{f\}$ is not a matching for any $f \in E \setminus M$. Let Z be the size of a maximum matching. Show that $|M| \geq \frac{1}{2}Z$ for any maximal matching M .

7

Branch and Bound

7.1 Divide and Conquer

Consider the problem:

$$Z = \max\{cx : x \in S\}.$$

How can we break the problem into a series of smaller problems that are easier, solve the smaller problems and then put the information together again to solve the original problem?

Proposition 7.1 *Let $S = S_1 \cup \dots \cup S_K$ be a decomposition of S into smaller sets, and let $Z^k = \max\{cx : x \in S_k\}$ for $k = 1, \dots, K$. Then $Z = \max_k Z^k$.*

A typical way to represent such a divide and conquer approach is via an enumeration tree. For instance, if $S \subseteq \{0, 1\}^3$, we might construct the enumeration tree shown in Figure 7.1.

Here we first divide S into $S_0 = \{x \in S : x_1 = 0\}$ and $S_1 = \{x \in S : x_1 = 1\}$, then $S_{00} = \{x \in S_0 : x_2 = 0\} = \{x \in S : x_1 = x_2 = 0\}$, $S_{01} = \{x \in S_0 : x_2 = 1\}$, and so on. Note that a leaf of the tree $S_{i_1 i_2 i_3}$ is nonempty if and only if $x = (i_1, i_2, i_3)$ is in S . Thus, the leaves of the tree correspond precisely to the points of $\{0, 1\}^3$ that we would examine if we carried out complete enumeration. Note that by convention the tree is drawn upside down with its root at the top.

Another example is the enumeration of all the tours of the traveling salesman problem. One way to divide up S , the set of all tours on four cities, is to take the union of the sets $S_{(12)}$, $S_{(13)}$, $S_{(14)}$ where $S_{(ij)}$ is the set of all tours containing arc (ij) . Then $S_{(12)}$ is divided again into $S_{(12)(23)}$ and $S_{(12)(24)}$, and so on. Note that at the first level, we have arbitrarily chosen to branch on the arcs leaving node 1, and at the second level on the arcs leaving node 2 that do not immediately create a subtour with the previous branching arc. The resulting tree is shown in Figure 7.2. Here the six leaves of the tree correspond to the $(n - 1)!$ tours shown, where $i_1 i_2 i_3 i_4$ means

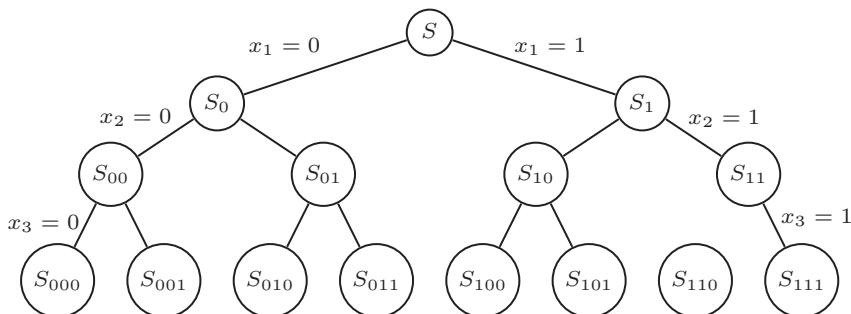


Figure 7.1 Binary enumeration tree.

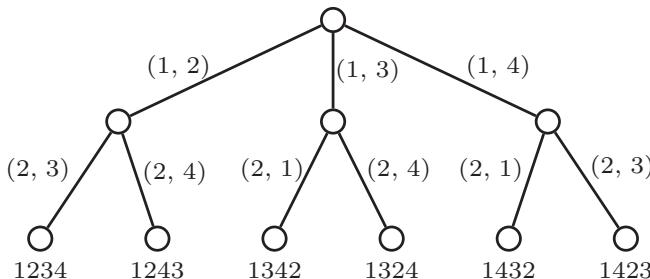


Figure 7.2 TSP enumeration tree.

that the cities are visited in the order i_1, i_2, i_3, i_4, i_1 respectively. Note that this is an example of multiway as opposed to binary branching, in which a set can be divided into more than two parts.

7.2 Implicit Enumeration

We saw in Chapter 1 that complete enumeration is totally impossible for most problems as soon as the number of variables in an integer program, or nodes in a graph exceeds 20 or 30. So we need to do more than just divide indefinitely. How can we use some bounds on the values of $\{Z^k\}$ intelligently? First, how can we put together bound information? Note that as we are maximizing, we take $\bar{Z}^k = -\infty$ when $S_k = \emptyset$ and $\underline{Z}^k = -\infty$ when no feasible solution in S_k has been found.

Proposition 7.2 *Let $S = S_1 \cup \dots \cup S_K$ be a decomposition of S into smaller sets, and let $Z^k = \max\{cx : x \in S_k\}$ for $k = 1, \dots, K$, \bar{Z}^k be an upper bound on Z^k and \underline{Z}^k be a lower bound on Z^k . Then $\bar{Z} = \max_k \bar{Z}^k$ is an upper bound on Z and $\underline{Z} = \max_k \underline{Z}^k$ is a lower bound on Z .*

Figure 7.3 Pruned by optimality.

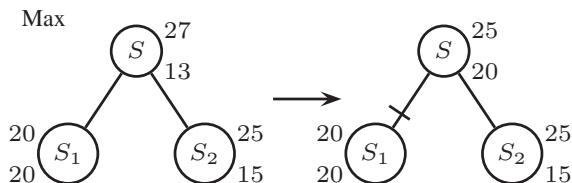
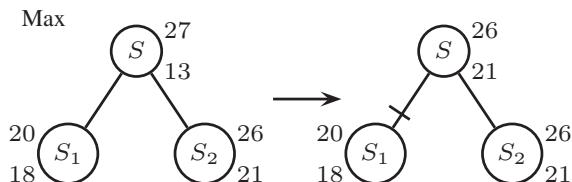


Figure 7.4 Pruned by bound.



Now, we examine three hypothetical examples to see how bound information, or partial information about a subproblem can be put to use. What can be deduced about lower and upper bounds on the optimal value Z and which sets need further examination in order to find the optimal value?

Example 7.1 In Figure 7.3, we show a decomposition of S into two sets S_1 and S_2 as well as upper and lower bounds on the corresponding problems.

We note first that $\bar{Z} = \max_k \bar{Z}^k = \max\{20, 25\} = 25$ and $\underline{Z} = \max_k \underline{Z}^k = \max\{20, 15\} = 20$. This is clearly an improvement on the initial lower and upper bounds of 13 and 27, respectively.

Second, we observe that as the lower and upper bounds on Z^1 are equal, $Z^1 = 20$, and there is no further reason to examine the set S_1 . Therefore, the branch S_1 of the enumeration tree can be *pruned by optimality*. \square

Example 7.2 In Figure 7.4, we again decompose S into two sets S_1 and S_2 and show upper and lower bounds on the corresponding problems.

We note first that $\bar{Z} = \max_k \bar{Z}^k = \max\{20, 26\} = 26$ and $\underline{Z} = \max_k \underline{Z}^k = \max\{18, 21\} = 21$.

Second, we observe that as the optimal value has value at least 21, and the upper bound $\bar{Z}^1 = 20$, no optimal solution can lie in the set S_1 . Therefore, the branch S_1 of the enumeration tree can be *pruned by bound*. \square

Example 7.3 In Figure 7.5, we again decompose S into two sets S_1 and S_2 with different upper and lower bounds.

We note first that $\bar{Z} = \max_k \bar{Z}^k = \max\{24, 37\} = 37$ and $\underline{Z} = \max_k \underline{Z}^k = \max\{13, -\infty\} = 13$. Here no other conclusion can be drawn and we need to explore both sets S_1 and S_2 further. \square

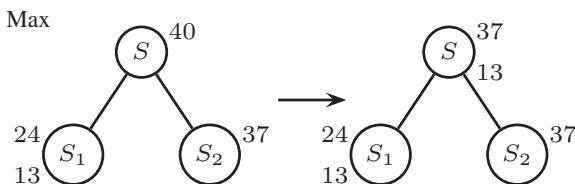


Figure 7.5 No pruning possible.

Based on these examples, we can list at least three reasons that allow us to prune the tree and thus enumerate a large number of solutions implicitly.

- (i) Pruning by optimality: $Z^t = \{\max cx : x \in S_t\}$ has been solved.
- (ii) Pruning by bound: $\bar{Z}^t \leq \underline{Z}$.
- (iii) Pruning by infeasibility: $S_t = \emptyset$.

If we now ask how the bounds are to be obtained, the reply is no different from in Chapter 2. The primal (lower) bounds are provided by feasible solutions and the dual (upper) bounds by relaxation or duality.

Building an implicit enumeration algorithm based on the above ideas is now in principle a fairly straightforward task. There are, however, many questions that must be addressed before such an algorithm is well defined. Some of the most important questions are

What relaxation or dual problem should be used to provide upper bounds? How should one choose between a fairly weak bound that can be calculated very rapidly and a stronger bound whose calculation takes a considerable time?

How should the feasible region be separated into smaller regions $S = S_1 \cup \dots \cup S_K$? Should we separate into two or more parts? Should we use a fixed a priori rule for dividing up the set, or should the divisions evolve as a function of the bounds and solutions obtained en route?

In what order should the subproblems be examined? Typically, there is a list of active problems that have not yet been pruned. Should the next one be chosen on the basis of last-in first-out, of best/largest upper bound first, or of some totally different criterion?

These and other questions will be discussed further once we have seen an example.

7.3 Branch and Bound: an Example

The most common way to solve integer programs is to use implicit enumeration, or *branch and bound*, in which linear programming relaxations provide the bounds. We first demonstrate the approach by an example.

Example 7.4

$$Z = \max 4x_1 - x_2 \quad (7.1)$$

$$7x_1 - 2x_2 \leq 14 \quad (7.2)$$

$$x_2 \leq 3 \quad (7.3)$$

$$2x_1 - 2x_2 \leq 3 \quad (7.4)$$

$$x \in \mathbb{Z}_+^2. \quad (7.5)$$

Bounding. To obtain a first upper bound, we add slack variables x_3, x_4, x_5 and solve the linear programming relaxation in which the integrality constraints are dropped. The resulting optimal basis representation is

$$\begin{aligned} \bar{Z} = \max & \frac{59}{7} & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 \\ & x_1 & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & = \frac{20}{7} \\ & x_2 & & +x_4 & = 3 \\ & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 = \frac{23}{7} \\ & x_1, x_2, x_3, x_4, x_5 & \geq 0. \end{aligned}$$

Thus, we obtain an upper bound $\bar{Z} = \frac{59}{7}$ and a nonintegral solution $(\bar{x}_1, \bar{x}_2) = (\frac{20}{7}, 3)$. Is there any straightforward way to find a feasible solution? Apparently not. As no feasible solution is yet available, we take as lower bound $\underline{Z} = -\infty$.

Branching. Now, because $\underline{Z} < \bar{Z}$, we need to divide or branch. How should we split up the feasible region? One simple idea is to choose an integer variable that is basic and fractional in the linear programming solution, and split the problem into two about this fractional value. If $x_j = \bar{x}_j \notin \mathbb{Z}^1$, one can take:

$$S_1 = S \cap \{x : x_j \leq \lfloor \bar{x}_j \rfloor\}$$

$$S_2 = S \cap \{x : x_j \geq \lceil \bar{x}_j \rceil\}.$$

It is clear that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$. Another reason for this choice is that the solution \bar{x} of LP(S) is not feasible in either LP(S_1) or LP(S_2). This implies that if there is no degeneracy (i.e. multiple optimal LP solutions), then $\max\{\bar{Z}_1, \bar{Z}_2\} < \bar{Z}$, so the upper bound will strictly decrease.

Following this idea, as $\bar{x}_1 = 20/7 \notin \mathbb{Z}^1$, we take $S_1 = S \cap \{x : x_1 \leq 2\}$ and $S_2 = S \cap \{x : x_1 \geq 3\}$. We now have the tree shown in Figure 7.6. The subproblems (nodes) that must still be examined are called *active*. Note that the new nodes immediately inherit the upper (dual) bound of their predecessor, so we have $\bar{Z}_1 = \bar{Z}_2 = \frac{59}{7}$.

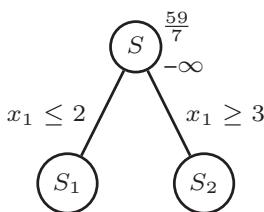


Figure 7.6 Partial branch-and-bound tree 1.

Choosing a Node. The list of active problems (nodes) to be examined now contains S_1, S_2 . We arbitrarily choose S_1 .

Reoptimizing. How should we solve the new modified linear programs $\text{LP}(S_i)$ for $i = 1, 2$ without starting again from scratch?

As we have just added one single upper or lower bound constraint to the linear program, our previous optimal basis remains dual feasible, and it is therefore natural to reoptimize from this basis using the dual simplex algorithm. Typically, only a few pivots will be needed to find the new optimal linear programming solution.

Applying this to the linear program $\text{LP}(S_1)$, we can write the new constraint $x_1 \leq 2$ as $x_1 + s = 2, s \geq 0$. Replacing the basic variable x_1 using the equation $x_1 + \frac{1}{7}x_3 + \frac{2}{7}x_4 = \frac{20}{7}$ permits us to rewrite the new constraint in terms of the nonbasic variables as

$$-\frac{1}{7}x_3 - \frac{2}{7}x_4 + s = -\frac{6}{7}.$$

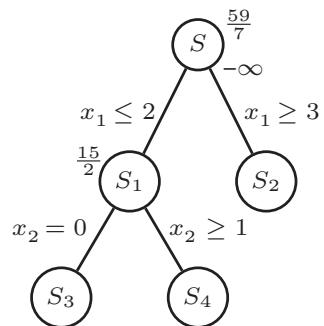
Thus, we have the dual feasible representation:

$$\begin{aligned} \bar{Z}_1 &= \max \frac{59}{7} & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 \\ & x_1 & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & = & \frac{20}{7} \\ & x_2 & & +x_4 & = & 3 \\ & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & = \frac{23}{7} \\ & & -\frac{1}{7}x_3 & -\frac{2}{7}x_4 & +s & = -\frac{6}{7} \\ & x_1, x_2, x_3, x_4, x_5, s & \geq 0. \end{aligned}$$

After two simplex pivots, the linear program is reoptimized, giving:

$$\begin{aligned} \bar{Z}_1 &= \max \frac{15}{2} & -\frac{1}{2}x_5 & -3s \\ & x_1 & & +s & = 2 \\ & x_2 & & -\frac{1}{2}x_5 & +s & = \frac{1}{2} \\ & x_3 & & -x_5 & -5s & = 1 \\ & x_4 & & +\frac{1}{2}x_5 & +6s & = \frac{5}{2} \\ & x_1, x_2, x_3, x_4, x_5, s & \geq 0 \end{aligned}$$

with $\bar{Z}_1 = \frac{15}{2}$, and $(\bar{x}_1^1, \bar{x}_2^1) = (2, \frac{1}{2})$.

Figure 7.7 Partial branch-and-bound tree 2.

Branching. S_1 cannot be pruned, so using the same branching rule as before, we create two new nodes $S_3 = S_1 \cap \{x : x_2 \leq 0\}$ and $S_4 = S_1 \cap \{x : x_2 \geq 1\}$, and add them to the node list. The tree is now as shown in Figure 7.7.

Choosing a Node. The active node list now contains S_2, S_3, S_4 . Arbitrarily choosing S_2 , we remove it from the node list and examine it in more detail.

Reoptimizing. To solve $\text{LP}(S_2)$, we use the dual simplex algorithm in the same way as above. The constraint $x_1 \geq 3$ is first written as $x_1 - t = 3, t \geq 0$, which expressed in terms of the nonbasic variables becomes

$$\frac{1}{7}x_3 + \frac{2}{7}x_4 + t = -\frac{1}{7}.$$

As $x_3, x_4, t \geq 0$, this equation cannot be satisfied. So the resulting linear program is infeasible, $\bar{Z}_2 = -\infty$, and node S_2 is pruned by infeasibility.

Choosing a Node. The node list now contains S_3, S_4 . Arbitrarily choosing S_4 , we remove it from the list.

Reoptimizing. $S_4 = S \cap \{x : x_1 \leq 2, x_2 \geq 1\}$. The resulting linear program has optimal solution $\bar{x}^4 = (2, 1)$ with value 7. As \bar{x}^4 is integer, $Z^4 = 7$.

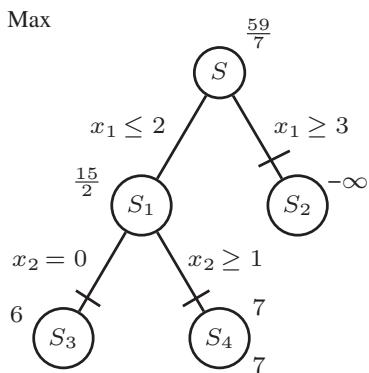
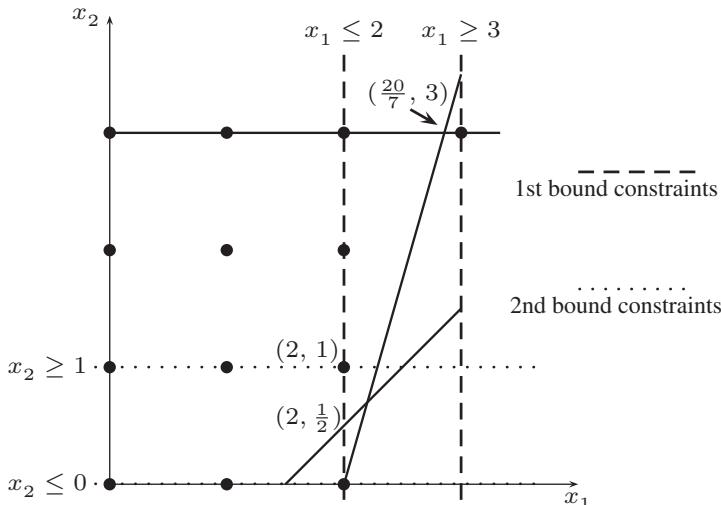
Updating the Incumbent. As the solution of $\text{LP}(S_4)$ is an integer, we update the value of the best feasible solution found $\underline{Z} \leftarrow \max\{\underline{Z}, 7\}$, and store the corresponding solution $(2, 1)$. S_4 is now pruned by optimality. Note that the value \underline{Z} is applicable at each node of the tree.

Choosing a Node. The node list now contains only S_3 .

Reoptimizing. $S_3 = S \cap \{x : x_1 \leq 2, x_2 \leq 0\}$. The resulting linear program has optimal solution $\bar{x}^3 = (\frac{3}{2}, 0)$ with value 6. As $\underline{Z} = 7 > \bar{Z}_3 = 6$, the node is *pruned by bound*.

Choosing a Node. As the node list is empty, the algorithm terminates. The incumbent solution $x = (2, 1)$ with value $Z = 7$ is optimal.

The complete branch-and-bound tree is shown in Figure 7.8. In Figure 7.9, we show graphically the feasible node sets S_i , the branching, the relaxations $\text{LP}(S_i)$, and the solutions encountered in the example. \square

**Figure 7.8** Complete branch-and-bound tree.**Figure 7.9** Division of the feasible region.

7.4 LP-Based Branch and Bound

In Figure 7.10, we present a flowchart of a simple branch and bound algorithm. Though not appearing in Example 7.4, the flowchart includes an initial heuristic step designed to find a good starting solution. We then discuss in more detail some of the practical aspects of developing and using such an algorithm.

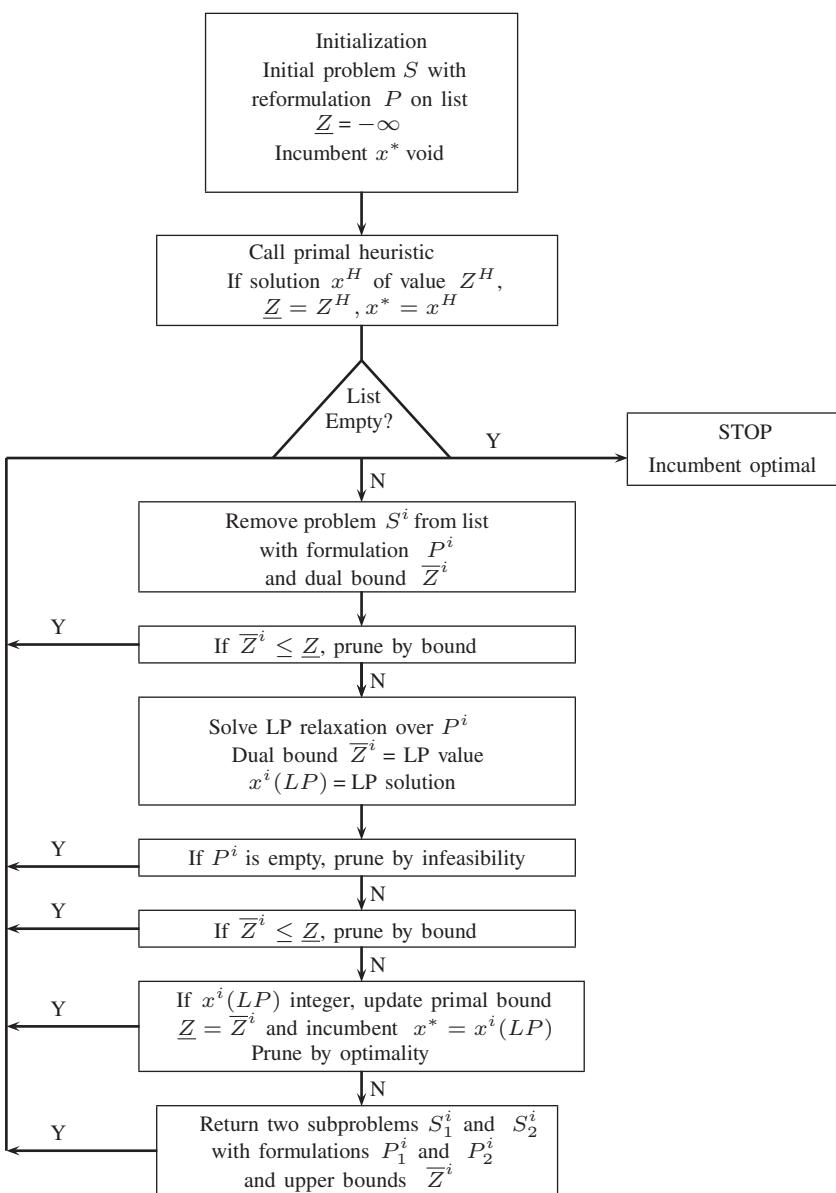


Figure 7.10 Branch-and-bound flow chart.

Storing the Tree

In practice we do not store a tree, but just the list of *active* nodes or subproblems that have not been pruned and that still need to be explored further. Here the question arises of how much information should be kept. Should we keep a minimum of information and be prepared to repeat certain calculations, or should we keep all the information available? At a minimum, the best-known dual bound and the variable lower and upper bounds needed to restore the subproblem are stored. Usually, an optimal or near-optimal basis is kept so that the linear programming relaxation can be reoptimized rapidly.

Returning to the questions raised earlier, there is no single answer that is best for all instances. The rules used are based on a combination of theory, common sense, and practical experimentation. In our example, the question of **how to bound** was solved by using an LP relaxation; **how to branch** was solved by choosing an integer variable that is fractional in the LP solution. However, as there is typically a choice of a set C of several candidates, we need a rule to choose between them. One common, but not very effective, choice is *the most fractional variable*:

$$\arg \max_{j \in C} \min[f_j, 1 - f_j],$$

where $f_j = x_j^* - \lfloor x_j^* \rfloor$, so that a variable with fractional value $f_j = \frac{1}{2}$ is best. Other rules, based on the idea of *estimating* the cost of forcing the variable x_j to become integer, are presented in the next section.

Choosing a node

This question was avoided in Example 7.4 by making an arbitrary choice. In practice, there are several contradictory arguments that can be invoked:

- (i) It is only possible to prune the tree significantly with a (primal) feasible solution, giving a hopefully good lower bound. Therefore, we should descend as quickly as possible in the enumeration tree to find a first feasible solution. This suggests the use of a *Depth-First Search* strategy. Another argument for such a strategy is the observation that it is always easy to resolve the linear programming relaxation when a simple constraint is added, and the optimal basis is available. Therefore, passing from a node to one of its immediate descendants is to be encouraged. In the example, this would imply that after treating node S_1 , the next node treated would be S_3 or S_4 rather than S_2 .
- (ii) To minimize the total number of nodes evaluated in the tree, the optimal strategy is to always choose the active node with the best (largest upper) bound (i.e. choose node s , where $\bar{Z}_s = \max_t \bar{Z}_t$). With such a rule, we will never divide any node whose upper bound \bar{Z}_t is less than the optimal value Z . This leads to a *Best Node First* strategy. In Example 7.4, this would imply that after treating

node S_1 , the next node chosen would be S_2 with bound $\frac{59}{7}$ from its predecessor, rather than S_3 or S_4 with bound $\frac{15}{2}$.

In practice, a compromise between these ideas is often adopted, involving an initial depth-first strategy until at least one feasible solution has been found, followed by a strategy mixing best node and depth first so as to try to prove optimality and also find better feasible solutions.

7.5 Using a Branch-and-Bound/Cut System

To go from a simple LP-based branch-and-bound algorithm to the highly impressive branch-and-cut algorithms that are now available, many important ideas and options have been introduced, some of which we now describe. The three most important are probably:

1. Preprocessing to clean up and tighten the model and possibly reduce its size (described in this chapter),
2. Cutting planes to improve the formulation in the course of the algorithm and provide better dual bounds (see Chapters 8 and 9),
3. Primal heuristics to find good primal feasible solutions and provide better primal bounds (see Chapter 13),

Other important options include:

- A choice of linear programming algorithms, primal or dual simplex, or an interior point algorithm
- A choice of pivot strategies for the simplex algorithms
- A choice of branching and node selection strategies including strong branching and pseudo-costs
- Special modeling functions: special ordered sets and semicontinuous variables
- Reduced cost fixing (see Exercise 7)
- User-selected priorities among the integer variables and among branching directions
- Symmetry breaking

We now describe in a little more detail these options, several of which can be modified by the user.

Simplex Strategies

Though the linear programming algorithms are finely tuned, the default strategy will not be best for all classes of problems. Different *simplex pricing* strategies may

make a huge difference in running times for a given class of models, so if similar models are resolved repeatedly or the linear programs seem very slow, some experimentation by the user with pricing strategies may be worthwhile. In addition, on very large models, *interior point methods* may be best for the solution of the first linear program. Unfortunately, up to now such methods are still not good for reoptimizing quickly at each node of the tree.

Branching and Node Selection

We assume a maximization problem. To choose a node from the node list, a natural idea is to try to estimate the optimal value that will be obtained from pursuing that branch. For an integer variable x_j taking a fractional value x_j^* , the *down-fractionality* is $f_j = x_j^* - \lfloor x_j^* \rfloor$ and the *up-fractionality* is $1 - f_j$. Now, let P_j^-, P_j^+ , called *pseudo-costs*, be estimates of the cost per unit of forcing x_j down, respectively up, to become integral. Then the estimated cost of setting $x_j \rightarrow \lfloor x_j^* \rfloor$ is the *down degradation* $D_j^- = f_j P_j^-$ and the cost of setting $x_j \rightarrow \lceil x_j^* \rceil$ is $D_j^+ = (1 - f_j) P_j^+$. Now, if Z_{LP} is the value of the LP solution at a node, one estimate of the value of that node is $Z_{LP} - \sum_j \min[D_j^-, D_j^+]$.

The question that arises is how to obtain the pseudocosts P_j^-, P_j^+ . Let Z_j^- be the cost when resolving the LP with $x_j = \lfloor x_j^* \rfloor$, with Z_j^+ defined analogously. Then natural estimates are $P_j^- = \frac{Z_{LP} - Z_j^-}{f_j}$ and $P_j^+ = \frac{Z_{LP} - Z_j^+}{1 - f_j}$. However, as calculating such estimates at each node is time-consuming, the same pseudo-cost values are used at different nodes of the tree and even then the values of Z_j^-, Z_j^+ may only be approximated by carrying out a few dual simplex pivots.

These values can also be used to choose a branching variable. Two possible criteria designed to choose a variable for which the degradations due to integrality are important are $\hat{j} = \operatorname{argmax}_j [(D_j^- + D_j^+)]$ or $\hat{j} = \operatorname{argmax}_j [\min(D_j^-, D_j^+)]$.

Strong Branching

The idea behind strong branching is that on difficult problems it should be worthwhile to do more work to try to choose a better branching variable. The system chooses a set C of basic integer variables that are fractional in the LP solution, branches up and down on each of them in turn, and reoptimizes on each branch either to optimality, or for a specified number of dual simplex pivots. Now, for each variable $j \in C$, it has upper bounds Z_j^D for the down-branch and Z_j^U for the up-branch. The variable having the largest effect (smallest dual bound)

$$j^* = \arg \min_{j \in C} \max[Z_j^D, Z_j^U]$$

is then chosen, and branching really takes place on this variable. Obviously, solving two LPs for each variable in C is costly, so such branching is either used selectively as a default or can be called by the user.

Special Ordered Sets

A special ordered set of type 1 (SOS1) is a set of variables x_1, \dots, x_k of which at most one can be positive. To see why such sets are interesting, consider the special SOS1 case, also called a *generalized upper bound* (GUB) constraint, in which $x \in \{0, 1\}^n$. Now, the set can be expressed explicitly in the form

$$\sum_{j=1}^k x_j = 1$$

with $x_j \in \{0, 1\}$ for $j = 1, \dots, k$. If the linear programming solution x^* has some of the variables x_1^*, \dots, x_k^* fractional, then the standard branching rule is to impose $S_1 = S \cap \{x : x_j = 0\}$ and $S_2 = S \cap \{x : x_j = 1\}$ for some $j \in \{1, \dots, k\}$. However, because of the SOS1 constraint, $\{x : x_j = 0\}$ leaves $k - 1$ possibilities $\{x : x_i = 1\}_{i \neq j}$, whereas $\{x : x_j = 1\}$ leaves only one possibility. So set S_1 is typically a much larger set than set S_2 and the tree is *unbalanced*.

SOS1 branching is designed to provide a more balanced division of S into S_1 and S_2 . Specifically, the user specifies an ordering of the variables in the SOS1 set j_1, \dots, j_k , and the branching scheme is then to set

$$\begin{aligned} S_1 &= S \cap \{x : x_{j_i} = 0 \text{ } i = 1, \dots, r\} \text{ and} \\ S_2 &= S \cap \{x : x_{j_i} = 0 \text{ } i = r + 1, \dots, k\}, \end{aligned}$$

where $r = \min\{t : \sum_{i=1}^t x_{j_i}^* \geq \frac{1}{2}\}$. In many cases, such a branching scheme is much more effective than the standard scheme of branching on a single variable and the number of nodes in the tree is significantly reduced.

SOS2 are sets of the form $\{\lambda \in \mathbb{R}_+^k : \sum_{i=1}^k \lambda_i = 1\}$ in which at most two variables can be nonzero and if two are nonzero they must be adjacent, i.e. x_j and x_{j+1} for some $j \in [1, k - 1]$. They are used among others to linearize one-dimensional nonlinear functions. In Figure 7.11, the nonlinear function $f : \mathbb{R}^1 \rightarrow \mathbb{R}^1$ is represented over the interval $[x^1, x^k]$ by the piecewise linear function \bar{f} with break points x^1, \dots, x^k . Using the linear constraints $x = \sum_{i=1}^k \lambda_i x^i$ and $\bar{f}(x) = \sum_{i=1}^k \lambda_i f(x^i)$ and imposing the SOS2 condition, we have that $x = \lambda_i x^i + (1 - \lambda_i)x_{i+1}$ with $0 < \lambda_i \leq 1$ for some i and $\bar{f}(x) = \lambda_i f(x^i) + (1 - \lambda_i)f(x^{i+1})$ as required. So separable functions of the form $f(x_1, \dots, x_n) = \sum_{j=1}^n f_j(x_j)$ can be linearized in this way.

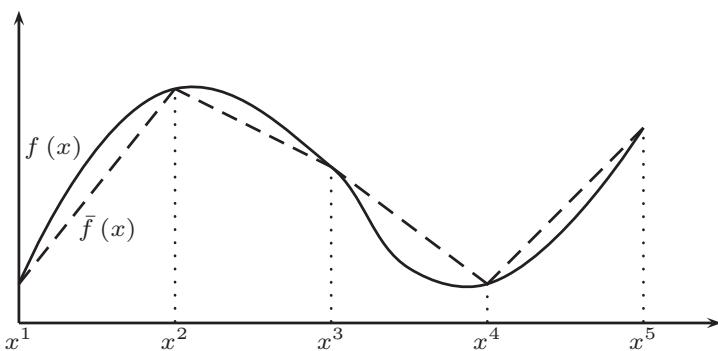


Figure 7.11 Piecewise linear approximation.

Semicontinuous Variables

If $x = 0$ or $\ell \leq x \leq u$ with $\ell > 0$, this can be modeled as a *semicontinuous variable* $\text{sc}(x, \ell, u)$. Note that if, at some stage in the tree, the lower bound on x is tightened to $\ell' > 0$, the semi-continuous variables can be replaced by a normal variable with $\max[\ell, \ell'] \leq x \leq u$. On the other hand, if the upper bound on x becomes $u' < \ell$, we can be set $x = 0$.

Indicator Variables

Suppose that a constraint $ax \leq b$ is active when some logical condition is satisfied, represented by an *indicator variable* $z \in \{0, 1\}$ taking the value $z = 1$. Certain MIP systems accept such variables and handle them directly, rather than introducing a big M constraint of the form $ax \leq b + M(1 - z)$ that can lead to numerical problems.

Priorities

Priorities allow the user to tell the system the relative importance of the integer variables. The user provides a file specifying a value (importance) of each integer variable. When it has to decide on a branching variable, the system will choose the highest priority integer variable whose current linear programming value is fractional. At the same time, the user can specify a preferred branching direction telling the system which of the two branches to explore first.

Symmetry Breaking/Orbital Branching

Many problem formulations contain symmetries, meaning that certain variables can be interchanged so that one feasible solution is converted into another feasible

solution of the same value. This is the case in graph coloring in which the colors can be interchanged, or in room assignments, when the rooms are identical. To give an example, consider the simple 0–1 integer program:

$$\begin{aligned} \max & 3 \sum_{j=1}^3 x_j + 2 \sum_{j=4}^6 x_j \\ & x_1 + x_2 + x_3 \leq 2 \\ & x_4 + x_5 + x_6 \geq 1 \\ & x_i + x_j + x_k \leq 2 \quad \text{for } 1 \leq i \leq 3, 4 \leq j, k \leq 6, j \neq k \\ & x \in \{0, 1\}^6. \end{aligned}$$

It is not difficult to see that variables $\{1, 2, 3\}$ are interchangeable and also variables $\{4, 5, 6\}$. What are the consequences when branching?

Before pursuing the example, we now formalize a little. We consider the problem $\max\{cx : Ax \leq b, x \in \{0, 1\}^n\}$.

Given a permutation π of the columns $\{1, \dots, n\}$ and a vector x_1, \dots, x_n , $\pi(x) = (x_{\pi(1)}, \dots, x_{\pi(n)})$. Note that π can also be represented by the n by n matrix P with $p_{i,\pi(i)} = 1$ for all i and $p_{ij} = 0$ otherwise.

Let $G(A, b, c)$ be the set of column permutations P for which there exists a row permutation R such that $RAP = A$, $Rb = b$, and $cP = c$.

Proposition 7.3 Suppose that $x \in \mathbb{R}_+^n$ satisfies $Ax \leq b$ and $\pi \in G(A, b, c)$, then $\pi(x)$ satisfies $Ax \leq b$ and $cx = c\pi(x)$.

Proof. Rewrite $Ax \leq b$ as $RAPx \leq Rb$, or $RA\pi(x) \leq Rb$. As multiplication by R is just a permutation of the rows, this is the same as $A\pi(x) \leq b$. \square

For a column j , let $\text{Orb}(j) = \{k : \pi(j) = k \text{ for some } \pi \in G(A, b, c)\}$

Proposition 7.4 At the root node, consider branching on x_i with $x_i = 1$ on the left branch and $x_i = 0$ on the right branch. If $j \in \text{Orb}(i)$, every solution \hat{x} on the right branch with $x_j = 1$ has an equivalent solution in the left branch. Thus, the right branch can be replaced by a branch with $x_j = 0$ for all $j \in \text{Orb}(i)$.

Proof. Take $i = 1$ and $j = 2$ wlog. The solution $\hat{x} = (0, 1, \hat{x}_3, \dots, \hat{x}_n)$ lies on the right branch. As $2 \in \text{Orb}(1)$, there is a permutation $\pi \in G(A, b, c)$ interchanging 1 and 2. Thus, the solution $(1, 0, \hat{x}_3, \dots, \hat{x}_n)$ is equivalent to \hat{x} . It lies in the left branch. So any solution with $x_2 = 1$ can be excluded on the right branch. \square

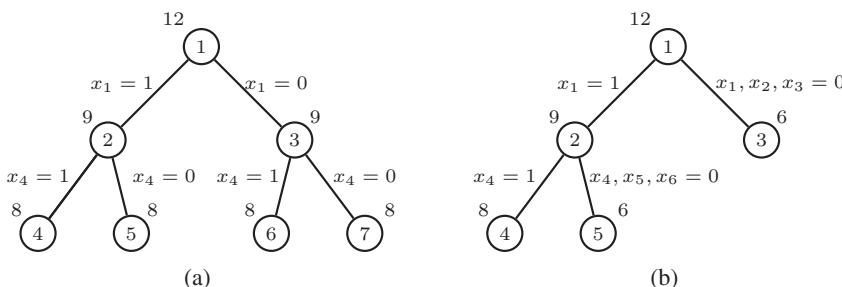


Figure 7.12 (a) Standard branching. (b) Orbital branching

Note that for a node further down the enumeration tree, variables that are fixed to 1 on the path will decrease the set of permutations and thus break up some of the orbits into smaller orbits.

Returning to the small 0–1 integer program above, $G(A, b, c)$ consists of all permutations of $\{1, 2, 3\}$ and all permutations of $\{4, 5, 6\}$. Thus, $\text{Orb}(1) = \text{Orb}(2) = \text{Orb}(3) = \{1, 2, 3\}$ and $\text{Orb}(4) = \text{Orb}(5) = \text{Orb}(6) = \{4, 5, 6\}$.

Using standard branching, the calculations are

$$\text{Node 1. } Z_{LP} = 12, x^* = \left(\frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}\right)$$

$$\text{Node 2. } Z_{LP} = 9, x^* = (1, 1, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$$

$$\text{Node 3. } Z_{LP} = 9, x^* = (0, 1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$$

Node 4. $Z_{LP} = 8, x^* = (1, 1, 0, 1, 0, 0)$. New incumbent $\underline{Z} = 8$. Prune node by optimality

Node 5. $Z_{LP} = 8, x^* = (1, 1, 0, 0, 1, 0)$. Prune node by optimality/bound

Node 6. $Z_{LP} = 8, x^* = (0, 1, 0, 1, 0, 0)$. Prune node by optimality/bound

Node 7. $Z_{LP} = 8, x^* = (0, 1, 1, 0, 0, 1)$. Prune node by optimality/bound

All nodes are pruned.

Using orbital branching, the calculations are

Nodes 1 and 2 are unchanged.

At node 3, $x_1 = x_2 = x_3 = 0$ as $x_j = 0$ for $j \in \text{Orb}(1)$. Now, $Z_{LP} = 6, x^* = (0, 0, 0, 1, 1, 1)$. Incumbent $\underline{Z} = 6$. Node pruned by optimality.

Node 4 is unchanged. New incumbent $\underline{Z} = 8$.

At node 5. For the set of solutions with $x_1 = 1$, $\text{Orb}(4) = \{4, 5, 6\}$ is unchanged, so we can add $x_4 = x_5 = x_6 = 0$. Now $Z_{LP} = 6, x^* = (1, 0, 1)$. Node pruned by optimality/bound.

All nodes are pruned. The corresponding trees are shown in Figure 7.12.

7.6 Preprocessing or Presolve

Before solving a linear or integer program, it is natural to check that the formulation is “sensible” and as strong as possible given the information available. All the commercial branch-and-cut systems carry out such a check, called *Preprocessing* or *Presolve*. The basic idea is to try to quickly detect and eliminate redundant constraints and variables, and tighten bounds where possible. Then if the resulting linear/integer program is smaller/tighter, it will typically be solved much more quickly. This is especially important in the case of branch-and-bound because tens or hundreds of thousands of linear programs may need to be solved.

Linear Programming Preprocessing

First, we demonstrate with an example.

Example 7.5 Consider the linear program

$$\begin{aligned} \max \quad & 2x_1 + x_2 - x_3 \\ \text{s.t. } & 5x_1 - 2x_2 + 8x_3 \leq 15 \\ & 8x_1 + 3x_2 - x_3 \geq 9 \\ & x_1 + x_2 + x_3 \leq 6 \\ & 0 \leq x_1 \leq 3 \\ & 0 \leq x_2 \leq 1 \\ & 1 \leq x_3. \end{aligned}$$

Tightening Bounds. Isolating variable x_1 in the first constraint, we obtain

$$5x_1 \leq 15 + 2x_2 - 8x_3 \leq 15 + 2 \times 1 - 8 \times 1 = 9,$$

where we use the bound inequalities $x_2 \leq 1$ and $-x_3 \leq -1$. Thus, we obtain the tightened bound $x_1 \leq \frac{9}{5}$.

Similarly, isolating variable x_3 , we obtain

$$8x_3 \leq 15 + 2x_2 - 5x_1 \leq 15 + 2 \times 1 - 5 \times 0 = 17,$$

and the tightened bound $x_3 \leq \frac{17}{8}$.

Isolating variable x_2 , we obtain

$$2x_2 \geq 5x_1 + 8x_3 - 15 \geq 5 \times 0 + 8 \times 1 - 15 = -7.$$

Here the existing bound $x_2 \geq 0$ is not changed.

Turning to the second constraint, isolating x_1 and using the same approach, we obtain $8x_1 \geq 9 - 3x_2 + x_3 \geq 9 - 3 + 1 = 7$, and an improved lower bound $x_1 \geq \frac{7}{8}$.

No more bounds are changed based on the second or third constraints. However, as certain bounds have been tightened, it is worth passing through the constraints again.

Isolating x_3 in constraint 1 now gives $8x_3 \leq 15 + 2x_2 - 5x_1 \leq 15 + 2 - 5 \times \frac{7}{8} = \frac{101}{8}$. Thus, we have the new bound $x_3 \leq \frac{101}{64}$.

Redundant Constraints. Using the latest upper bounds in constraint 3, we see that

$$x_1 + x_2 + x_3 \leq \frac{9}{5} + 1 + \frac{101}{64} < 6,$$

and so this constraint is redundant and can be discarded. The problem is now reduced to

$$\begin{array}{lllll} \max & 2x_1 & +x_2 & -x_3 \\ & 5x_1 & -2x_2 & +8x_3 & \leq 15 \\ & 8x_1 & +3x_2 & -x_3 & \geq 9 \\ \frac{7}{8} \leq x_1 & \leq \frac{9}{5}, & 0 \leq x_2 \leq 1, & 1 \leq x_3 \leq \frac{101}{64}. \end{array}$$

Variable Fixing (by Duality). Considering variable x_2 , observe that increasing its value makes all the constraints (other than its bound constraints) less tight. As the variable has a positive objective coefficient, it is advantageous to make the variable as large as possible, and thus set it to its upper bound. Another way to arrive at a similar conclusion is to write out the LP dual. For the dual constraint corresponding to the primal variable x_2 to be feasible, namely $-2u_1 + 3u_2 + 1u_3 \geq 1$ with $u_1, u_3 \geq 0, u_2 \leq 0$, the dual variable u_3 associated with the constraint $x_2 \leq 1$ must be positive. This implies by complementary slackness that $x_2 = 1$ in any optimal solution.

Similarly, decreasing x_3 makes the constraints less tight. As the variable has a negative objective coefficient, it is best to make it as small as possible, and thus set it to its lower bound $x_3 = 1$. Finally, the LP is reduced to the trivial problem:

$$\max \left\{ 2x_1 : \frac{7}{8} \leq x_1 \leq \frac{9}{5} \right\}.$$

□

Formalizing the above ideas is straightforward.

Proposition 7.5 Consider the set $S = \{x : a_0x_0 + \sum_{j=1}^n a_jx_j \leq b, l_j \leq x_j \leq k_j \text{ for } j = 0, 1, \dots, n\}$.

(i) *Bounds on Variables.* If $a_0 > 0$, then

$$x_0 \leq \left(b - \sum_{j \geq 1 : a_j > 0} a_j l_j - \sum_{j \geq 1 : a_j < 0} a_j k_j \right) / a_0,$$

and if $a_0 < 0$, then

$$x_0 \geq \left(b - \sum_{j \geq 1 : a_j > 0} a_j l_j - \sum_{j \geq 1 : a_j < 0} a_j k_j \right) / a_0.$$

(ii) *Redundancy.* The constraint $a_0 x_0 + \sum_{j=1}^n a_j x_j \leq b$ is redundant if

$$\sum_{j : a_j > 0} a_j k_j + \sum_{j : a_j < 0} a_j l_j \leq b.$$

(iii) *Infeasibility.* $S = \emptyset$ if

$$\sum_{j : a_j > 0} a_j l_j + \sum_{j : a_j < 0} a_j k_j > b.$$

(iv) *Variable Fixing.* For a maximization problem in the form: $\max\{cx : Ax \leq b, l \leq x \leq k\}$, if $a_{ij} \geq 0$ for all $i = 1, \dots, m$ and $c_j < 0$, then $x_j = l_j$. Conversely, if $a_{ij} \leq 0$ for all $i = 1, \dots, m$ and $c_j > 0$, then $x_j = k_j$.

Integer Programming Preprocessing

Turning now to integer programming problems, preprocessing can sometimes be taken a step further. Obviously, if $x_j \in \mathbb{Z}^1$ and the bounds l_j or k_j are not integer, we can tighten to

$$\lceil l_j \rceil \leq x_j \leq \lfloor k_j \rfloor.$$

For mixed integer programs with variable upper and lower bound constraints $l_j x_j \leq y_j \leq k_j x_j$ with $x_j \in \{0, 1\}$, it is also important to use the tightest bound information.

Generating Logical Inequalities

For 0–1 integer programs, it is common to look for simple “logical” or “Boolean” constraints involving only one or two variables, and then either add them to the problem or use them to fix some variables. Again, we demonstrate by example.

Example 7.6 Consider the set of constraints involving four 0–1 variables:

$$\begin{aligned} 7x_1 + 3x_2 - 4x_3 - 2x_4 &\leq 1 \\ -2x_1 + 7x_2 + 3x_3 + x_4 &\leq 6 \\ -2x_2 - 3x_3 - 6x_4 &\leq -5 \\ 3x_1 - 2x_3 &\geq -1 \\ x &\in \{0, 1\}^4. \end{aligned}$$

Examining row 1, we see that if $x_1 = 1$, then necessarily $x_3 = 1$, and similarly $x_1 = 1$ implies $x_4 = 1$. This can be formulated with the linear inequalities $x_1 \leq x_3$ and $x_1 \leq x_4$. We see also that the constraint is infeasible if both $x_1 = x_2 = 1$ leading to the constraint $x_1 + x_2 \leq 1$.

Row 2 gives the inequalities $x_2 \leq x_1$ and $x_2 + x_3 \leq 1$.

Row 3 gives $x_2 + x_4 \geq 1$ and $x_3 + x_4 \geq 1$.

Row 4 gives $x_1 \geq x_3$.

Combining Pairs of Logical Inequalities. We consider pairs involving the same variables.

From rows 1 and 4, we have $x_1 \leq x_3$ and $x_1 \geq x_3$, which together give $x_1 = x_3$.

From rows 1 and 2, we have $x_1 + x_2 \leq 1$ and $x_2 \leq x_1$ which together give $x_2 = 0$. Now, from $x_2 + x_4 \geq 1$ and $x_2 = 0$, we obtain $x_4 = 1$.

Simplifying. Making the substitutions $x_2 = 0, x_3 = x_1, x_4 = 1$, all four constraints of the feasible region are redundant, and we are left with $x_1 \in \{0, 1\}$, so the only feasible solutions are $(1, 0, 1, 1)$ and $(0, 0, 0, 1)$. \square

To formalize these ideas, it is useful to consider the complement variable of a 0–1 variable z_j , namely $\bar{z}_j = 1 - z_j$. Now, observe that any 0–1 knapsack set: $\sum_{j=1}^n a'_j z_j \leq b'$, $z \in \{0, 1\}^n$ can be rewritten as $\sum_{j=1}^n a_j x_j \leq b$, $x \in \{0, 1\}^n$, where $a_j = |a'_j|$ for all j , $x_j = z_j$ if $a'_j > 0$, $x_j = \bar{z}_j$ if $a'_j < 0$ and $b = b' - \sum_{j: a'_j < 0} a'_j$. Note that here $a_j \geq 0$ for all $j = 1, \dots, n$.

Proposition 7.6 *Given a 0–1 knapsack set in the form $X = \{x \in \{0, 1\}^n : \sum_{j=1}^n a_j x_j \leq b\}$ with $a_j \geq 0$ for all j ,*

- (i) $X = \emptyset$ if $b < 0$.
 - (ii) $X = \{0, 1\}^n$ if $\sum_{j=1}^n a_j \leq b$. In other words, the constraint is redundant.
 - (iii) If $a_k > b$, then $x_k = 0$ in all feasible solutions.
 - (iv) If $a_j + a_k > b$ with $j \neq k$, then $x_j + x_k \leq 1$ in all feasible solutions.
- Combining Pairs of Logical Inequalities*
- (v) If $x_j + x_k \leq 1$ and $x_j + \bar{x}_k \leq 1$ ($x_j \leq x_k$), then $x_j = 0$.
 - (vi) If $x_j + x_k \leq 1$ and $\bar{x}_j + \bar{x}_k \leq 1$ ($x_j + x_k \geq 1$), then $x_j + x_k = 1$.
 - (vii) If $x_j + \bar{x}_k \leq 1$ and $\bar{x}_j + \bar{x}_k \leq 1$, then $x_k = 1$.

The logical inequalities can also be viewed as providing a foretaste of the valid inequalities to be developed in Chapter 8.

Clique Finding

First we consider an example:

Suppose that we have found the three inequalities in 0, 1 variables $x_2 + x_4 \leq 1$, $x_2 \leq x_7$, and $x_4 \leq x_7$. These can be replaced by the stronger constraint $x_2 + x_4 \leq x_7$. One way to see this is to rewrite the last two constraints as $x_2 + \bar{x}_7 \leq 1$ and $x_4 + \bar{x}_7 \leq 1$. Now, the three constraints be viewed as representing edge constraints in a stable set problem on the nodes representing x_2 , x_4 , and \bar{x}_7 . As the edges form a clique, this gives the clique constraint $x_2 + x_4 + \bar{x}_7 \leq 1$.

To generalize, construct a graph with $2n$ nodes representing each variable x_i and its complement \bar{x}_i for all i . Add an edge between nodes x_i and x_j when $x_i + x_j \leq 1$, an edge between x_i and \bar{x}_j whenever $x_i \leq x_j$ and an edge between \bar{x}_i and \bar{x}_j whenever $x_i + x_j \geq 1$. Find a maximal clique $C = (C^+, C^-)$ in the graph, where C^+ are the nodes corresponding to a variable x_i and C^- to variables \bar{x}_i . This gives the valid clique constraint

$$\sum_{i \in C^+} x_i + \sum_{j \in C^-} (1 - x_j) \leq 1.$$

Strengthening Inequalities

Here we first state a simple result that can be used to strengthen certain inequalities before giving an example.

Proposition 7.7 Consider a constraint $w + ax \geq b$ where $w \in \mathbb{R}_+^1$ and $x \in \mathbb{Z}_+^1$.

- (i) If $a > b > 0$, the constraint can be replaced by the tighter constraint, $w + bx \geq b$.
- (ii) Consider a constraint $w + ax \leq b$, where $w \leq W$ and $x \leq k \in \mathbb{Z}^1$, $x \in \mathbb{Z}^1$. If $a > \alpha = W + ka - b > 0$, the constraint can be replaced by

$$w + ax \leq W + \alpha(k - 1).$$

Proof.

- (i) A point $(w, x) \in \mathbb{R}_+^1 \times \mathbb{Z}_+^1$ is feasible for the two variants of the constraint if and only if either $w \geq b$ or $x \geq 1$.
- (ii) If the constraint is nonredundant, $\alpha = W + ka - b > 0$. Setting $\bar{w} = W - w$ and $\bar{x} = k - x$, we obtain $\bar{w} + a\bar{x} \geq W + ka - b$ with $\bar{w} \in \mathbb{R}_+^1$ and $\bar{x} \in \mathbb{Z}_+^1$. By (i), this can be replaced by $\bar{w} + a\bar{x} \geq \alpha$, or in terms of the original variables $w + (W + ka - b)x \leq W + (W + ka - b)(k - 1)$. \square

Example 7.7 Consider a constraint $6y_1 - 11y_2 - x_1 + 12x_2 \leq 54$ with bounds $y_1 \leq 2, y_2 \geq -1, x_1 \geq 3, x_2 \leq 3, x \in \mathbb{Z}^2$. Now, using the bounds, $w = 6y_1 - 11y_2 - x_1 \leq 12 + 11 - 3 = 20$. So $W = 20$, $a = 12$ and $\alpha = 2$. Setting $\bar{w} = 20 - w$ and $\bar{x}_2 = 3 - x_2$, we obtain: $\bar{w} + 12\bar{x}_2 \geq 2$ with $\bar{w} \in \mathbb{R}_+^1, \bar{x}_2 \in \mathbb{Z}_+^1$. By (i) of Proposition 7.7, this can be replaced by $\bar{w} + 2\bar{x}_2 \geq 2$, or in terms of the original variables

$$6y_1 - 11y_2 - x_1 + 2x_2 \leq 24.$$

□

Numerous other tests, such as probing or searching for disconnected components, have been incorporated into the systems and are carried out automatically, so that the user has in most cases little or no reason to intervene.

7.7 Notes

- 7.2 The first paper presenting a branch-and-bound algorithm for integer programming is by Land and Doig [201]. The two-way branching scheme commonly used is from Dakin [88]. Little et al. [211] present a computationally successful application to the TSP problem using an assignment relaxation. Balas [20] developed an algorithm for 0–1 problems using simple tests to obtain dual bounds and check primal feasibility.
 - 7.4 Almost all mixed integer programming codes since the 1960s have been linear programming based branch-and-bound codes. Around the year 2000, violated valid inequalities were added automatically at the top node, leading to so-called *cut-and-branch* and when cuts were also added elsewhere in the tree, the name *branch-and-cut* became common. More references are given in the notes for Chapter 9.
 - 7.5 An early discussion of important elements of commercial codes can be found in Beale [36]. GUB/SOS branching is from Beale and Tomlin [37], probing from Guignard and Spielberg [169], and strong branching from Applegate et al. [13]. Constraint branching was used for TSP problems in Clochard and Naddef [72] and by Cook et al. [81] in their implementation of basis reduction for integer programming based upon the fundamental paper of Lenstra [206]. Experiments with various branch-and-bound strategies are reported in Linderoth and Savelsbergh [209] and more recently in Morrison et al. [231]. Branching rules are studied in Achterberg et al. [6].
- As solving linear programs forms such an important part of an integer programming algorithm, improvements in solving linear programs are crucial. All recent commercial codes include an interior point algorithm, as for many large linear programs, the latter algorithm is faster than the simplex method.

However, because reoptimization with the simplex method is easier than with interior point codes, the simplex method is still used in branch-and-bound. Improving reoptimization with interior point codes is still a major challenge. For recent work on the role of interior point algorithms, see Bertold et al. [47]. Using the symmetry groups to break symmetries was introduced in Margot [221] and the use of orbits for branching in Margot [222]. In [223], he discusses in detail the different ways that have been proposed to break symmetries. Orbital branching is from Ostrowski [237] and Ostrowski et al. [238]. For a recent computational study of different symmetry-breaking methods, see Pfetsch and Rehn [249].

Knapsack problems, in which the linear programming relaxations can be solved by inspection, have always been treated by specialized codes; see the books by Martello and Toth [224] and Kellerer et al. [195].

- 7.6 Preprocessing is crucially important for the rapid solution of linear programs. Its importance for integer programs is recognized in Brearley et al. [61] and Savelsbergh [264]. The progress reports on MIP, see Bixby et al. [54] and Achterberg and Wunderling [7] show the crucial role of preprocessing and Achterberg et al. [5] contains a detailed description of the Presolve routines implemented in the commercial system Gurobi [171], including inequality strengthening and special ordered set reductions. Many other preprocessing steps requiring more work are also described such as identification of parallel rows and dominating columns, combining rows to obtain more zero coefficients, probing, simple lifting, etc. See also Gamrath et al. [132]. Clique finding is presented in Atamturk et al. [17].

7.8 Exercises

1. Consider the enumeration tree (minimization problem) in Figure 7.13:
 - (i) Give tightest possible lower and upper bounds on the optimal value z .
 - (ii) Which nodes can be pruned and which must be explored further?
2. Consider the two-variable integer program:

$$\begin{aligned} \max \quad & 9x_1 + 5x_2 \\ \text{s.t. } & 4x_1 + 9x_2 \leq 35 \\ & x_1 \leq 6 \\ & x_1 - 3x_2 \geq 1 \\ & 3x_1 + 2x_2 \leq 19 \\ & x \in Z_+^2. \end{aligned}$$

Solve by branch-and-bound graphically and algebraically.

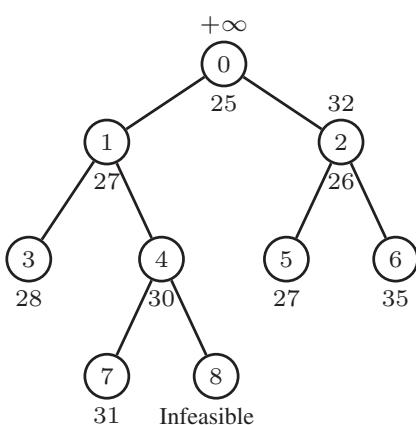


Figure 7.13 Enumeration tree (min).

3. Consider the 0–1 knapsack problem:

$$\max \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x \in \{0, 1\}^n \right\}$$

with $a_j, c_j > 0$ for $j = 1, \dots, n$.

- (i) Show that if $\frac{c_1}{a_1} \geq \dots \geq \frac{c_n}{a_n} > 0$, $\sum_{j=1}^{r-1} a_j \leq b$ and $\sum_{j=1}^r a_j > b$, the solution of the LP relaxation is $x_j = 1$ for $j = 1, \dots, r - 1$, $x_r = (b - \sum_{j=1}^{r-1} a_j)/a_r$, and $x_j = 0$ for $j > r$.
- (ii) Solve the instance

$$\max 17x_1 + 10x_2 + 25x_3 + 17x_4$$

$$5x_1 + 3x_2 + 8x_3 + 7x_4 \leq 12$$

$$x \in \{0, 1\}^4$$

by branch-and-bound.

4. Solve the integer knapsack problem:

$$\max 10x_1 + 12x_2 + 7x_3 + 2x_4$$

$$4x_1 + 5x_2 + 3x_3 + 1x_4 \leq 10$$

$$x_1, x_2 \in Z_+^1, x_3, x_4 \in \{0, 1\}$$

by branch-and-bound.

5. (i) Solve the STSP instance with $n = 5$ and distance matrix

$$(c_e) = \begin{pmatrix} - & 10 & 2 & 4 & 6 \\ - & - & 9 & 3 & 1 \\ - & - & - & 5 & 6 \\ - & - & - & - & 2 \end{pmatrix}$$

by branch-and-bound using a 1-tree relaxation (see Definition 2.3) to obtain bounds.

- (ii) Solve the TSP instance with $n = 4$ and distance matrix

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

by branch-and-bound using an assignment relaxation to obtain bounds.

- (iii) Describe clearly the branching rules you use in (i) and (ii), and motivate your choice.

6. Using a branch-and-cut system, solve your favorite integer program with different choices of branching and node selection rules, and report on the differences in the running time and the number of nodes in the enumeration tree.
7. *Reduced cost fixing.* Suppose that the linear programming relaxation of an integer program has been solved to optimality, and the objective function is then represented in the form

$$Z = \max cx, cx = \bar{a}_{00} + \sum_{j \in NB_1} \bar{a}_{0j}x_j + \sum_{j \in NB_2} \bar{a}_{0j}(x_j - u_j),$$

where NB_1 are the nonbasic variables at zero, and NB_2 are the nonbasic variables at their upper bounds u_j , $\bar{a}_{0j} \leq 0$ for $j \in NB_1$, and $\bar{a}_{0j} \geq 0$ for $j \in NB_2$. In addition suppose that a primal feasible solution of value \underline{Z} is known. Prove the following: In any optimal solution,

$$x_j \leq \left\lfloor \frac{\bar{a}_{00} - \underline{Z}}{-\bar{a}_{0j}} \right\rfloor \text{ for } j \in NB_1, \text{ and}$$

$$x_j \geq u_j - \left\lceil \frac{\bar{a}_{00} - \underline{Z}}{\bar{a}_{0j}} \right\rceil \text{ for } j \in NB_2.$$

8. Compare the branch-and-bound trees for the instance

$$\min\{x_1 : x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 = 3, x \in \{0, 1\}^5\}$$

using standard and orbital branching.

9. Consider the 0–1 problem:

$$\begin{aligned} \max \quad & 5x_1 - 7x_2 - 10x_3 + 3x_4 - 5x_5 \\ & x_1 + 3x_2 - 5x_3 + x_4 + 4x_5 \leq 0 \\ & -2x_1 - 6x_2 + 3x_3 - 2x_4 - 2x_5 \leq -4 \\ & 2x_2 - 2x_3 - x_4 + x_5 \leq -2 \\ & x \in \{0, 1\}^5. \end{aligned}$$

Simplify using logical inequalities.

10. Prove Proposition 7.6 concerning 0–1 preprocessing.

- 11*. Some small integer programs are very difficult for mixed integer programming systems. Try to find a feasible solution to the integer equality knapsack: $\{x \in \mathbb{Z}_+^n : \sum_{j=1}^n a_j x_j = b\}$ with $a = (12\ 228, 36\ 679, 36\ 682, 48\ 908, 61\ 139, 73\ 365)$ and $b = 89\ 716\ 837$.

12. Suppose that $P^i = \{x \in \mathbb{R}^n : A^i x \leq b^i\}$ for $i = 1, \dots, m$ and that $C_k \subseteq \{1, \dots, m\}$ for $k = 1, \dots, K$. A *disjunctive program* is a problem of the form

$$\max\{cx : x \in \cup_{i \in C_k} P^i \text{ for } k = 1, \dots, K\}.$$

Show how the following can be modeled as disjunctive programs:

- (i) a 0–1 integer program.
- (ii) a linear complementarity problem: $w = q + Mz$, $w, z \in \mathbb{R}_+^m$, $w_j z_j = 0$ for $j = 1, \dots, m$.
- (iii) a single machine sequencing problem with job processing times p_j , and variables t_j representing the start time of job j for $j = 1, \dots, n$.
- (iv) the nonlinear expression $z = \max\{3x_1 + 2x_2 - 3, 9x_1 - 4x_2 + 6\}$.
- (v) the constraint: if $x \in \mathbb{R}_+^1$ is positive, then x lies between 20 and 50, and is a multiple of 5.

13. Devise a branch-and-bound algorithm for a disjunctive program.

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 $X = \{(x, y) \in Z_+^n \times \mathbb{R}_+^p : Ax + Gy \leq b\}$ 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 $X \subseteq \mathbb{R}^n$ if $\pi x \leq \pi_0$ for all $x \in X$.

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

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.

Companion website: www.wiley.com/go/wolsey/integerprogramming2e

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 7.6 in looking at preprocessing.

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

$$X = \{x \in \{0, 1\}^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2\}.$$

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. \square

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

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

It is easily checked that the inequality

$$y \leq 5x$$

is valid because $X = \{(0, 0), (1, y) \text{ with } 0 \leq y \leq 5\}$. 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 $y \leq 5x$ gives us the convex hull of X .

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

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

All feasible solutions satisfy $y_{ij} \leq b_j x_j$ and $y_{ij} \leq a_i$ with $x_j \in \{0, 1\}$. This is precisely the situation above leading to the family of valid inequalities $y_{ij} \leq \min\{a_i, b_j\}x_j$. \square

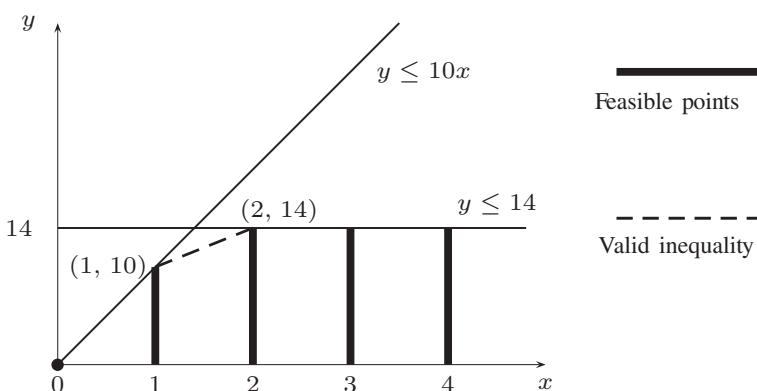


Figure 8.1 Mixed integer inequality.

Example 8.3 A Mixed Integer Set. Consider the set

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

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

$$X = \{(x, y) : y \leq Cx, 0 \leq y \leq b, x \in \mathbb{Z}_+^1\},$$

it suffices to add the valid inequality $y \leq b - \gamma(K - x)$ where $K = \lceil \frac{b}{C} \rceil$ and $\gamma = b - (\lceil \frac{b}{C} \rceil - 1)C$. \square

Example 8.4 A Combinatorial Set: Stable Sets. See Exercise 2 in Chapter 2 for definitions. Consider the set X of incidence vectors of stable sets in a graph $G = (V, E)$:

$$\begin{aligned} x_i + x_j &\leq 1 \quad \text{for } (i, j) \in E \\ x &\in \{0, 1\}^{|V|}. \end{aligned}$$

Take a clique $U \subseteq V$. As there is an edge between every pair of nodes in U , any stable set contains at most one node of U . Therefore,

$$\sum_{j \in U} x_j \leq 1$$

is a valid inequality for X . \square

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

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

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 an integer, and all the coefficients are integers, the lhs must be an 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 :

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

□

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, with x_{ij} denoting the number of trucks of type i serving client j , is

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

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 $X = P \cap (\mathbb{Z}^4 \times \mathbb{R}^1)$, where

$$P = \{(x, s) \in \mathbb{R}_+^4 \times \mathbb{R}_+^1 : 13x_1 + 20x_2 + 11x_3 + 6x_4 + s \geq 72\}.$$

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

$$\frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq \frac{72-s}{11},$$

suggesting that there is a valid inequality

$$2x_1 + 2x_2 + x_3 + x_4 + \alpha y \geq 7 \quad \text{for some } \alpha. \quad (8.1)$$

Looking at the rhs term $\frac{72-y}{11}$, we see that the rhs $\lceil \frac{72-y}{11} \rceil$ decreases from 7 to 6 at the critical value $y = 6$, 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:

$$\frac{4}{3}x_1 + 2x_2 + x_3 + x_4 + \frac{1}{6}y \geq 7.$$

□

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).

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 $\min\{ub : uA - v = \pi, u \geq 0, v \geq 0\} \leq \pi_0$. □

Valid Inequalities for Integer Programs

Now, we consider the feasible region of an integer program:

$$\{x : Ax \leq b, x \in \mathbb{Z}_+^n\}$$

and ask the same question.

Surprisingly, the complete answer is in some sense given in the following very simple observation:

Proposition 8.3 Let $X = \{x \in \mathbb{Z}^1 : x \leq b\}$, then the inequality $x \leq \lfloor b \rfloor$ is valid for X .

We have already used this idea in Example 8.5. We now give two more examples.

Example 8.7 Valid Inequalities for Matching. Here we give an alternative algebraic justification for the validity of the blossom inequality (4.3) that can be broken up into three steps.

- (i) Take a nonnegative linear combination of the constraints, the degree constraints $\sum_{e \in \delta(i)} x_e \leq 1$ 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 $x \in \mathbb{Z}^n$, the lhs $\sum_{e \in E(T)} x_e$ must be an integer, and so we can replace the rhs value by the largest integer less than or equal to the rhs value. So

$$\sum_{e \in E(T)} x_e \leq \left\lfloor \frac{|T|}{2} \right\rfloor$$

is a valid inequality. □

Example 8.8 Valid Inequalities for an Integer Program. An identical approach can be used to derive valid inequalities for any integer programming region. Let $X = P \cap \mathbb{Z}^n$ be the set of integer points in P , where P is given by

$$7x_1 - 2x_2 \leq 14$$

$$x_2 \leq 3$$

$$2x_1 - 2x_2 \leq 3$$

$$x \geq 0.$$

- (i) Combining the constraints with nonnegative weights $u = \left(\frac{2}{7}, \frac{37}{63}, 0 \right)$, we obtain the valid inequality for P

$$2x_1 + \frac{1}{63}x_2 \leq \frac{121}{21}.$$

- (ii) As $x \geq 0$, the coefficients on the lhs can be reduced to the nearest integer giving the valid inequality for P :

$$2x_1 + 0x_2 \leq \frac{121}{21}.$$

- (iii) Now as the lhs is integral for all points of X , we can reduce the rhs to the nearest integer, and we obtain the valid inequality for X :

$$2x_1 \leq \left\lfloor \frac{121}{21} \right\rfloor = 5.$$

Observe that if we repeat the procedure, and use a weight of $\frac{1}{2}$ on this last constraint, we obtain the tighter inequality $x_1 \leq \lfloor \frac{5}{2} \rfloor = 2$. \square

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 $X = P \cap \mathbb{Z}^n$, where $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$, A is an $m \times n$ matrix with columns $\{a_1, a_2, \dots, a_n\}$ and $u \in \mathbb{R}_+^m$:

- (i) the inequality

$$\sum_{j=1}^n ua_j x_j \leq ub$$

is valid for P as $u \geq 0$ and $\sum_{j=1}^n a_j x_j \leq b$,

- (ii) the inequality

$$\sum_{j=1}^n \lfloor ua_j \rfloor x_j \leq ub$$

is valid for P as $x \geq 0$,

- (iii) the inequality

$$\sum_{j=1}^n \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor$$

is valid for X as x is integer, and thus $\sum_{j=1}^n \lfloor ua_j \rfloor x_j$ is integer.

The surprising fact is that this simple procedure is sufficient to generate all valid inequalities for an integer program.

Theorem 8.1 *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 $P = \{x \in \mathbb{R}^n : Ax \leq b, \mathbf{0} \leq x \leq \mathbf{1}\} \neq \emptyset$, $X = P \cap \mathbb{Z}^n$, 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 *Chvátal–Gomory (C-G) inequality*, or in other words that it can be obtained by applying the Chvátal–Gomory procedure a finite number of times. \square

Claim 1 The inequality $\pi x \leq \pi_0 + t$ is a valid inequality for P for some $t \in \mathbb{Z}_+^1$.

Proof. $Z_{LP} = \max\{\pi x : x \in P\}$ is bounded as P is bounded. Take $t = \lceil Z_{LP} \rceil - \pi_0$. \square

Claim 2 There exists a sufficiently large integer M that the inequality

$$\pi x \leq \pi_0 + M \sum_{j \in N^0} x_j + M \sum_{j \in N^1} (1 - x_j) \quad (8.2)$$

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 $x^* \in \mathbb{Z}^n$, then the inequality $\pi x \leq \pi_0$ is satisfied, and so (8.2) 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 nonintegral extreme points x^* of P . Taking $M \geq \frac{t}{\alpha}$, it follows that for all extreme points of P ,

$$\pi x^* \leq \pi_0 + t \leq \pi_0 + M \sum_{j \in N^0} x_j^* + M \sum_{j \in N^1} (1 - x_j^*). \quad \square$$

Claim 3 If $\pi x \leq \pi_0 + \tau + 1$ is a C-G inequality for X with $\tau \in \mathbb{Z}_+^1$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in N^0} x_j + \sum_{j \in N^1} (1 - x_j) \quad (8.3)$$

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 (8.2) with weight $1/M$. The resulting C-G inequality is (8.3). \square

Claim 4 If

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0 \cup \{p\}} x_j + \sum_{j \in T^1} (1 - x_j) \quad (8.4)$$

and

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1 \cup \{p\}} (1 - x_j) \quad (8.5)$$

are C-G inequalities for X where (T^0, T^1) is any partition of $\{1, \dots, p - 1\}$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1} (1 - x_j) \quad (8.6)$$

is a C-G inequality for X .

Proof. Take the inequalities (8.4) and (8.5) with weights $1/2$ and the resulting C-G inequality is (8.6). \square

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 $\{1, \dots, p-1\}$. \square

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. \square

For inequalities generated by other arguments, it is sometimes interesting to see how easily they are generated as C-G inequalities, see Exercise 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 $X = P \cap \mathbb{Z}^n$, 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 $X = P' \cap \mathbb{Z}^n$. Then we can apply our favorite algorithm, branch-and-cut or whatever, to formulation P' .

Advantages. We can use standard branch-and-cut 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-cut 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 we 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-cut software because there are too many constraints.

How can we 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 $X^2 = P^2 \cap \mathbb{Z}^n$ is easy. Then we may be able to find an explicit description of $P'^2 = \text{conv}(P^2 \cap \mathbb{Z}^n)$. 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, we 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 8.9 Uncapacitated Facility Location. Take the “weak” formulation used in the 1950s and 1960s:

$$\begin{aligned} \sum_{j=1}^n y_{ij} &= 1 \quad \text{for } i = 1, \dots, m \\ \sum_{i=1}^m y_{ij} &\leq mx_j \quad \text{for } j = 1, \dots, n \\ y_{ij} &\geq 0 \quad \text{for } i = 1, \dots, m, j = 1, \dots, n \\ 0 \leq x_j &\leq 1 \quad \text{for } j = 1, \dots, n. \end{aligned}$$

Let X_j be the set of points in the polyhedron P_j :

$$\begin{aligned} \sum_{i=1}^m y_{ij} &\leq mx_j \\ 0 \leq y_{ij} &\leq 1 \quad \text{for } i = 1, \dots, m \\ 0 \leq x_j &\leq 1, \end{aligned}$$

with x_j integer. The convex hull P'_j of the set X_j is given by

$$\begin{aligned} y_{ij} &\leq x_j \quad \text{for } i = 1, \dots, m \\ y_{ij} &\geq 0 \quad \text{for } i = 1, \dots, m \\ 0 \leq x_j &\leq 1. \end{aligned}$$

Now, the reformulation obtained by replacing P_j by P'_j for $j = 1, \dots, n$ is the “strong” formulation P' :

$$\begin{aligned} \sum_{j=1}^n y_{ij} &= 1 \quad \text{for } i = 1, \dots, m \\ y_{ij} &\leq x_j \quad \text{for } i = 1, \dots, m, j = 1, \dots, n \\ y_{ij} &\geq 0 \quad \text{for } i = 1, \dots, m, j = 1, \dots, n \\ 0 \leq x_j &\leq 1 \quad \text{for } j = 1, \dots, n. \end{aligned}$$

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. \square

Example 8.10 Constant Capacity Lot-Sizing. Using the same basic formulation as in Example 6.2, the feasible region X is given by

$$\begin{aligned}s_{t-1} + y_t &= d_t + s_t \quad \text{for } t = 1, \dots, n \\ y_t &\leq Cx_t \quad \text{for } t = 1, \dots, n \\ s_0 = s_n &= 0, s \in \mathbb{R}_+^{n+1}, y \in \mathbb{R}^n, x \in \{0, 1\}^n.\end{aligned}$$

We derive two families of inequalities for X . First consider any point $(x, s, y) \in X$. As $s_{t-1} \geq 0$, the inequality $y_t \leq d_t + s_t$ clearly holds. Along with $y_t \leq Cx_t$ and $x_t \in \{0, 1\}$, it is then not difficult to show that $y_t \leq d_t x_t + s_t$ 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 $\sum_{i=1}^t y_i \geq \sum_{i=1}^t d_i$. Then using $y_i \leq Cx_i$ gives $C \sum_{i=1}^t x_i \geq \sum_{i=1}^t d_i$, or $\sum_{i=1}^t x_i \geq (\sum_{i=1}^t d_i) / C$. Now, as $\sum_{i=1}^t x_i$ is integral, we can use Chvátal–Gomory integer rounding to obtain the valid inequality

$$\sum_{i=1}^t x_i \geq \left\lceil \left(\sum_{i=1}^t d_i \right) / C \right\rceil.$$

Adding just these $2n$ inequalities significantly strengthens the formulation of this problem. \square

8.5 Automatic Reformulation or Cutting Plane Algorithms

Suppose that $X = P \cap \mathbb{Z}^n$ 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, the goal is not to find the complete convex hull, but to find a good approximation to $\text{conv}(X)$ 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:

$$\bar{Z}^t = \max\{cx : x \in P^t\}.$$

Let x^t be an optimal solution.

If $x^t \in \mathbb{Z}^n$, stop. x^t is an optimal solution for IP.

If $x^t \notin \mathbb{Z}^n$, 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-cut solver. 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 Section 8.6, 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:

$$\begin{aligned} \max \bar{a}_{00} + \sum_{j \in NB} \bar{a}_{0j} x_j \\ x_{B_u} + \sum_{j \in NB} \bar{a}_{uj} x_j = \bar{a}_{u0} \quad \text{for } u = 1, \dots, m \\ x \geq 0 \text{ and integer} \end{aligned} \tag{8.7}$$

with $\bar{a}_{0j} \leq 0$ when $x_j \in NB$ 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 $\bar{a}_{u0} \notin \mathbb{Z}^1$. Choosing such a row, the Chvátal–Gomory cut for row u is

$$x_{B_u} + \sum_{j \in NB} \lfloor \bar{a}_{uj} \rfloor x_j \leq \lfloor \bar{a}_{u0} \rfloor. \tag{8.8}$$

Rewriting this inequality by eliminating x_{B_u} gives

$$\sum_{j \in NB} (\bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor) x_j \geq \bar{a}_{u0} - \lfloor \bar{a}_{u0} \rfloor$$

or

$$\sum_{j \in NB} f_{uj} x_j \geq f_{u0}, \quad (8.9)$$

where $f_{uj} = \bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor$ for $j \in NB$, and $f_{u0} = \bar{a}_{u0} - \lfloor \bar{a}_{u0} \rfloor$.

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 lhs and rhs of the Chvátal–Gomory inequality (8.8), and hence also of (8.9), is integral when x is integral, so that when (8.9) 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 8.11 Consider the integer program

$$\begin{aligned} Z = \max \quad & 4x_1 - x_2 \\ & 7x_1 - 2x_2 \leq 14 \\ & x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x_1, \quad x_2 \geq 0 \text{ and integer.} \end{aligned}$$

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 the equivalent representation:

$$\begin{aligned} Z = \max \frac{59}{7} & -\frac{4}{7}x_3 - \frac{1}{7}x_4 \\ x_1 & +\frac{1}{7}x_3 + \frac{2}{7}x_4 = \frac{20}{7} \\ x_2 & +x_4 = 3 \\ & -\frac{2}{7}x_3 + \frac{10}{7}x_4 + x_5 = \frac{23}{7} \\ x_1, x_2, x_3, x_4, x_5 & \geq 0 \text{ and integer.} \end{aligned}$$

The optimal linear programming solution is $x = \left(\frac{20}{7}, 3, 0, 0, \frac{23}{7}\right) \notin Z_+^5$, 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

$$\begin{array}{lll}
 Z = \max \frac{15}{2} & -\frac{1}{2}x_5 & -3s \\
 x_1 & +s & = 2 \\
 x_2 & -\frac{1}{2}x_5 & +s = \frac{1}{2} \\
 x_3 & -x_5 & -5s = 1 \\
 x_4 & +\frac{1}{2}x_5 & +6s = \frac{5}{2} \\
 x_1, x_2, x_3, x_4, x_5, s & \geq 0 \text{ and integer.}
 \end{array}$$

Now, the new optimal linear programming solution $x = \left(2, \frac{1}{2}, 1, \frac{5}{2}, 0\right)$ 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

$$\begin{array}{lll}
 Z = \max 7 & -3s & -t \\
 x_1 & +s & = 2 \\
 x_2 & +s & -t = 1 \\
 x_3 & -5s & -2t = 2 \\
 x_4 & +6s & +t = 2 \\
 x_5 & & -t = 1 \\
 x_1, x_2, x_3, x_4, x_5, s, t & \geq 0 \text{ and integer.}
 \end{array}$$

Now, the linear programming solution is integral and optimal, so $(x_1, x_2) = (2, 1)$ solves the original integer program. \square

It is natural to also look at the cuts in the space of the original variables.

Example 8.11 (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 $\left(\frac{20}{7}, 3\right)$.

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. \square

A general formula for the cut in terms of the original variables is given by

Proposition 8.4 Let β be row u of B^{-1} , and $q_i = \beta_i - \lfloor \beta_i \rfloor$ 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

$$\sum_{j=1}^n \lfloor qa_j \rfloor x_j \leq \lfloor qb \rfloor.$$

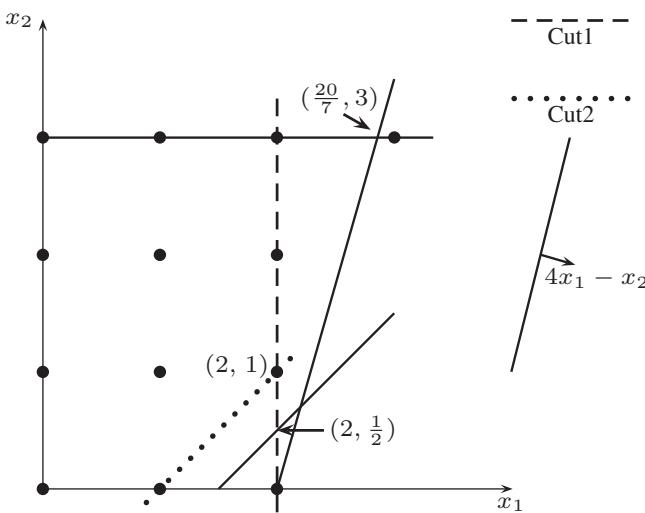


Figure 8.2 Gomory cutting planes.

Looking at the first Gomory cut generated in Example 8.11, β is given by the coefficients of the slack variables in row $u = 1$, so $\beta = \left(\frac{1}{7}, \frac{2}{7}, 0\right)$. Thus, $q = \left(\frac{1}{7}, \frac{2}{7}, 0\right)$, and we obtain $1x_1 + 0x_2 \leq \lfloor \frac{20}{7} \rfloor = 2$.

8.7 Mixed Integer Cuts

8.7.1 The Basic Mixed Integer Inequality

We saw above that when $x \leq b$, $x \in \mathbb{Z}^1$, the rounding inequality $x \leq \lfloor b \rfloor$ 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 8.5 Let $X^\geq = \{(x, y) \in \mathbb{Z}_+^1 \times \mathbb{R}^1 : x + y \geq b\}$, and $f = b - \lfloor b \rfloor > 0$. The inequality

$$y \geq f(\lceil b \rceil - x) \quad \text{or} \quad \frac{y}{f} + x \geq \lceil b \rceil$$

is valid for X^\geq .

Proof. If $x \geq \lceil b \rceil$, then $y \geq 0 \geq f(\lceil b \rceil - x)$. If $x \leq \lfloor b \rfloor$, then

$$\begin{aligned} y &\geq b - x = f + (\lceil b \rceil - x) \\ &\geq f + f(\lceil b \rceil - x), \quad \text{as } \lceil b \rceil - x \geq 0 \text{ and } f < 1, \\ &= f(\lceil b \rceil - x). \end{aligned}$$

□

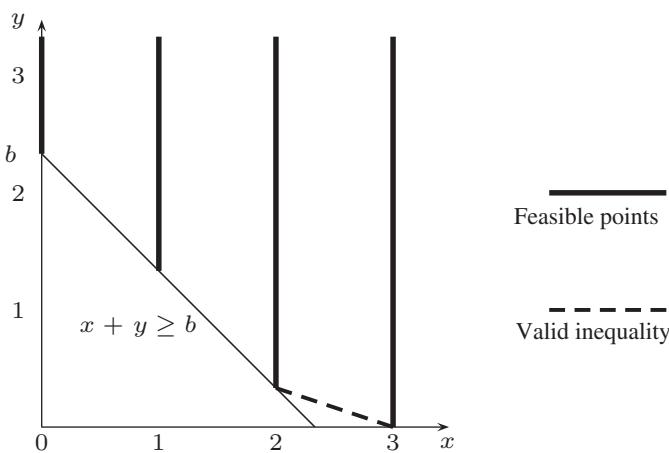


Figure 8.3 The basic mixed integer inequality.

The situation is shown in Figure 8.3. The following corollary allows us to compare more directly with the all-integer case.

Corollary 8.1 If $X^{\leq} = \{(x, y) \in \mathbb{Z}_+^1 \times \mathbb{R}^1 : x \leq b + y\}$ and $f = b - \lfloor b \rfloor > 0$, the inequality

$$x \leq \lfloor b \rfloor + \frac{y}{1-f}$$

is valid for X^{\leq} .

Proof. Rewriting $x \leq b + y$ as $y - x \geq -b$ and observing that $-b - \lceil -b \rceil = 1 - f$, we obtain from Proposition 8.5 that $\frac{y}{1-f} - x \geq \lceil -b \rceil = -\lceil b \rceil$. \square

Thus, we see that when the continuous variable $y = 0$, we obtain the basic integer rounding inequality.

Example 8.6 (cont.) The trucking example discussed earlier led to the set $2x_1 + 2x_2 + x_3 + x_4 + \frac{y}{11} \geq \frac{72}{11}$ with $x \in \mathbb{Z}_+^4$ and $y \geq 0$. Using Proposition 8.5 with $\lceil b \rceil = 7$ and $f = \frac{6}{11}$, we obtain immediately that

$$\frac{y}{11} \geq \frac{6}{11}(7 - 2x_1 - 2x_2 - x_3 - x_4)$$

is a valid inequality. \square

8.7.2 The Mixed Integer Rounding (MIR) Inequality

To obtain a slight variant of the basic inequality, we consider a set

$$X^{\text{MIR}} = \{(x, y) \in \mathbb{Z}_+^2 \times \mathbb{R}_+^1 : a_1 x_1 + a_2 x_2 \leq b + y\},$$

where a_1 , a_2 , and b are scalars with $b \notin \mathbb{Z}^1$.

Proposition 8.6 *Let $f = b - \lfloor b \rfloor$ and $f_i = a_i - \lfloor a_i \rfloor$ for $i = 1, 2$. Suppose $f_1 \leq f < f_2$, then*

$$\lfloor a_1 \rfloor x_1 + \left(\lfloor a_2 \rfloor + \frac{f_2 - f}{1 - f} \right) x_2 \leq \lfloor b \rfloor + \frac{y}{1 - f} \quad (8.10)$$

is valid for X^{MIR} .

Proof. $(x, y) \in X^{\text{MIR}}$ satisfies $\lfloor a_1 \rfloor x_1 + \lfloor a_2 \rfloor x_2 \leq b + y + (1 - f_2)x_2$ as $x_1 \geq 0$, and $a_2 = \lfloor a_2 \rfloor - (1 - f_2)$. Now the Corollary 8.1 to Proposition 8.5 gives

$$\lfloor a_1 \rfloor x_1 + \lfloor a_2 \rfloor x_2 \leq \lfloor b \rfloor + [y + (1 - f_2)x_2]/(1 - f),$$

which is the required inequality. \square

Example 8.12 Consider the set $X = \{(x, y) \in \mathbb{Z}_+^3 \times \mathbb{R}_+^1 : \frac{10}{3}x_1 + 1x_2 + \frac{11}{4}x_3 \leq \frac{21}{2} + y\}$. Using Proposition 8.6, we have $f = 1/2$, $f_1 = 1/3$, $f_2 = 0$, $f_3 = 3/4$, and thus

$$3x_1 + x_2 + \frac{5}{2}x_3 \leq 10 + 2y$$

is valid for X . \square

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:

$$X^G = \left\{ (x_{B_u}, x, y) \in \mathbb{Z}^1 \times \mathbb{Z}_+^{n_1} \times \mathbb{R}_+^{n_2} : x_{B_u} + \sum_{j \in N_1} \bar{a}_{uj} x_j + \sum_{j \in N_2} \bar{a}_{uj} y_j = \bar{a}_{u0} \right\},$$

where $n_i = |N_i|$ for $i = 1, 2$.

Proposition 8.7 *If $\bar{a}_{u0} \notin \mathbb{Z}^1$, $f_j = \bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor$ for $j \in N_1$, and $f_0 = \bar{a}_{u0} - \lfloor \bar{a}_{u0} \rfloor$, the Gomory mixed integer cut*

$$\sum_{j \leq f_0} f_j x_j + \sum_{j > f_0} \frac{f_0(1 - f_j)}{1 - f_0} x_j + \sum_{\bar{a}_{uj} > 0} \bar{a}_{uj} y_j - \sum_{\bar{a}_{uj} < 0} \frac{f_0}{1 - f_0} \bar{a}_{uj} y_j \geq f_0$$

is valid for X^G .

Proof. The mixed integer rounding inequality (8.10) for X^G is

$$x_{B_u} + \sum_{f_j \leq f_0} \lfloor \bar{a}_{uj} \rfloor x_j + \sum_{f_j > f_0} \left(\lfloor \bar{a}_{uj} \rfloor + \frac{f_j - f_0}{1 - f_0} \right) x_j + \sum_{\bar{a}_{uj} < 0} \frac{\bar{a}_{uj}}{1 - f_0} y_j \leq \lfloor \bar{a}_{u0} \rfloor.$$

Substituting for x_{B_u} proves the claim. \square

Example 8.13 Consider the mixed integer program:

$$\begin{aligned} Z = \min \quad & 4x_1 + 7x_2 + y_1 - y_2 \\ & 5x_1 + 4x_2 - y_1 - 2y_2 = 5 \\ & 3x_2 + y_1 - y_2 = 6 \\ x_1 \in \mathbb{Z}_+^2, y \in \mathbb{R}_+^2. \end{aligned}$$

Solving as a linear program gives

$$\begin{aligned} Z = \min \quad & \frac{86}{7} + \frac{8}{7}x_1 + \frac{12}{7}y_2 \\ & \frac{5}{7}x_1 + x_2 - \frac{3}{7}y_2 = \frac{11}{7} \\ & \frac{-15}{7}x_1 + y_1 + \frac{2}{7}y_2 = \frac{9}{7} \\ x_1 \in \mathbb{Z}_+^2, y \in \mathbb{R}_+^2. \end{aligned}$$

The basic variable x_2 is fractional and the first row gives the mixed integer rounding (MIR) cut

$$\frac{1}{3}x_1 + x_2 - y_2 \leq 1$$

which after elimination of x_2 becomes the Gomory mixed integer cut:

$$\frac{8}{21}x_1 + \frac{4}{7}y_2 \geq \frac{4}{7}.$$

Adding this cut and reoptimizing leads to the solution $x = (0, 2)$, $y = (1, 1)$ which is feasible and hence optimal for the mixed integer program. \square

8.7.4 Split Cuts

An alternative viewpoint on both MIR inequalities and Gomory mixed integer cuts is provided by split cuts.

Definition 8.2 The inequality $cx + hy \leq c_0$ is a *split cut* for $P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ if there exists $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}^1$ such that $cx + hy \leq c_0$ is valid for the sets $P \cap \{(x, y) : \pi x \leq \pi_0\}$ and $P \cap \{(x, y) : \pi x \geq \pi_0 + 1\}$.

As $P \cap (\mathbb{Z}^n \times \mathbb{R}^p) \subseteq (P \cap \{(x, y) : \pi x \leq \pi_0\}) \cup (P \cap \{(x, y) : \pi x \geq \pi_0 + 1\})$, we have immediately:

Proposition 8.8 If $cx + hy \leq c_0$ is a split cut for $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$, then $cx + hy \leq c_0$ is a valid inequality for X .

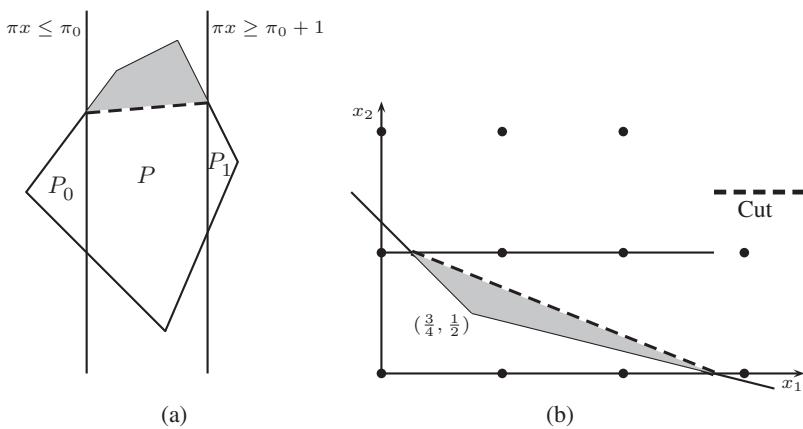


Figure 8.4 Two split cuts. The region cut off is shaded.

Then from LP duality:

Corollary 8.2 $cx + hy \leq c_0$ is a split cut for $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$ if there exists $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}^1$ and $\alpha, \beta \geq 0$ such that $cx + hy \leq c_0 + \alpha(\pi x - \pi_0)$ and $cx + hy \leq c_0 + \beta(\pi_0 + 1 - \pi x)$ are valid inequalities for P .

In Figure 8.4a, we show an example of a split cut. In fact, it is not difficult to show that MIR, split, and Gomory mixed integer cuts are essentially equivalent in the sense that any inequality that is of one type, can also be obtained as an inequality of the other two types.

Example 8.14 Consider the two variable set $X = P \cap \mathbb{Z}^2$, where $P = \{x \in \mathbb{R}_+^2 : x_1 + 4x_2 \geq \frac{11}{4}, x_1 + x_2 \geq \frac{5}{4}\}$. The two constraints intersect in the vertex $x = \left(\frac{3}{4}, \frac{1}{2}\right) \notin \mathbb{Z}^2$ of P . Consider the split $x_2 \leq 0$ and $x_2 \geq 1$. The first constraint intersects $x_2 = 0$ in the point $\left(\frac{11}{4}, 0\right)$ and the second $x_2 = 1$ in the point $\left(\frac{1}{4}, 1\right)$. Joining together these two points gives the split cut $4x_1 + 10x_2 \geq 11$.

In the terms of Corollary 8.2, the inequality $4x_1 + 16x_2 \geq 11$ is valid for P and can be rewritten in the form $4x_1 + 10x_2 \geq 11 - 6x_2$ and the inequality $4x_1 + 4x_2 \geq 5$ is valid for P and can be rewritten in the form $4x_1 + 10x_2 \geq 11 - 6(1 - x_2)$. See Figure 8.4b. \square

Now, given a point $(x^*, y^*) \in P \setminus X$ and a split $(\pi, \pi_0) \in \mathbb{Z}^n \times \mathbb{Z}^1$ with $\pi_0 < \pi x^* < \pi_0 + 1$, the natural question is how to find a valid inequality for the union of the two sets produced by the split that cuts off (x^*, y^*) . This is the subject of Lift and Project discussed in Section 8.8.

8.8 Disjunctive Inequalities and Lift-and-Project*

The set $X = X^1 \cup X^2$ with $X^i \subseteq \mathbb{R}_+^n$ 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 Propositions 8.5 and 8.8 in deriving the basic mixed integer inequality and the split cut.

Proposition 8.9 *If $\sum_{j=1}^n \gamma_j^i x_j \leq \gamma_0^i$ is valid for X^i for $i = 1, 2$, then the inequality*

$$\sum_{j=1}^n \gamma_j x_j \leq \gamma_0$$

is valid for X if $\gamma_j \leq \min[\gamma_j^1, \gamma_j^2]$ for $j = 1, \dots, n$ and $\gamma_0 \geq \max[\gamma_0^1, \gamma_0^2]$.

Proof. If $x \in X$, then $x \in X^1$ or $x \in X^2$. If $x \in X^i$, then as $x \geq 0$, $\sum_{j=1}^n \gamma_j x_j \leq \sum_{j=1}^n \gamma_j^i x_j \leq \gamma_0^i \leq \gamma_0$ for $i = 1, 2$. Thus, the inequality is valid for all $x \in X$. \square

Disjunctions of polyhedra are particularly interesting. We have already seen an example in the case of split cuts. Modeling such sets is easy (see Exercise 3 in Chapter 1 and Exercise 12).

Proposition 8.10 *If $P^i = \{x \in \mathbb{R}^n : A^i x \leq b^i\}$ for $i = 1, 2$ are bounded nonempty polyhedra, then $\text{conv}(P^1 \cup P^2) =$*

$$\{x : x = y^1 + y^2, A^i y^i \leq b^i z^i \text{ for } i = 1, 2, z^1 + z^2 = 1, z^1, z^2 \geq 0\}.$$

Using Proposition 8.2, it is also easy to characterize valid inequalities for such disjunctions.

Proposition 8.11 *If $P^i = \{x \in \mathbb{R}_+^n : A^i x \leq b^i\}$ for $i = 1, 2$ are nonempty polyhedra, then (γ, γ_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 $\gamma \leq u^i A^i$ and $\gamma_0 \geq u^i b^i$ for $i = 1, 2$.*

Example 8.15 Let $P^1 = \{x \in \mathbb{R}_+^2 : -x_1 + x_2 \leq 1, x_1 + x_2 \leq 5\}$ and $P^2 = \{x \in \mathbb{R}_+^2 : x_2 \leq 4, -2x_1 + x_2 \leq -6, x_1 - 3x_2 \leq -2\}$. Taking $u^1 = (2, 1)$ and $u^2 = \left(\frac{5}{2}, \frac{1}{2}, 0\right)$ and applying Proposition 8.11 gives that $-x_1 + 3x_2 \leq 7$ is valid for $P^1 \cup P^2$. See Figure 8.5. \square

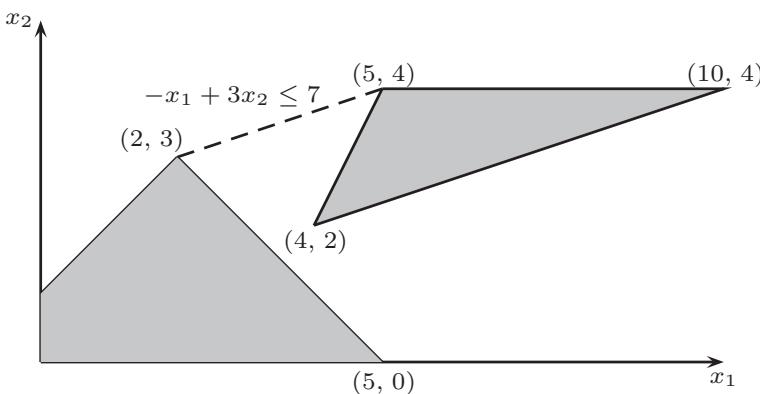


Figure 8.5 Disjunctive inequality.

Lift-and-Project

Here we present results for 0–1 problems. They extend easily to mixed 0–1 problems. Take $X = P \cap \mathbb{Z}^n$ with $P = \{x : Ax \leq b, \mathbf{0} \leq x \leq \mathbf{1}\}$. Let $P^0 = P \cap \{x \in \mathbb{R}^n : x_j = 0\}$, and $P^1 = P \cap \{x \in \mathbb{R}^n : x_j = 1\}$ for some $j \in \{1, \dots, n\}$.

Proposition 8.12 *The inequality (γ, γ_0) is a valid for $\text{conv}(P^0 \cup P^1)$ if there exists $u^i \in \mathbb{R}_+^m, v^i \in \mathbb{R}_+^n, w^i \in \mathbb{R}_+^1$ for $i = 0, 1$ such that*

$$\begin{aligned}\gamma &\leq u^0 A + v^0 + w^0 e_j, \quad \gamma \leq u^1 A + v^1 - w^1 e_j, \\ \gamma_0 &\geq u^0 b + \mathbf{1} \cdot v^0, \quad \gamma_0 \geq u^1 b + \mathbf{1} \cdot v^1 - w^1.\end{aligned}$$

Proof. Apply Proposition 8.11 with $P^0 = \{x \in \mathbb{R}_+^n : Ax \leq b, x \leq \mathbf{1}, x_j \leq 0\}$ and $P^1 = \{x \in \mathbb{R}_+^n : Ax \leq b, x \leq \mathbf{1}, -x_j \leq -1\}$. \square

Example 8.16 Consider the 0–1 knapsack problem

$$\begin{aligned}\max \quad & 12x_1 + 14x_2 + 7x_3 + 12x_4 \\ \text{s.t.} \quad & 4x_1 + 5x_2 + 3x_3 + 6x_4 \leq 8 \\ & x \in \{0, 1\}^4\end{aligned}$$

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 (γ, γ_0) given by Proposition 8.11. To do this, we solve a linear program consisting of maximizing $\gamma x^* - \gamma_0$ over the polyhedron

describing the coefficients of the valid inequalities given in the proposition, namely

$$\max 1.0\gamma_1 + 0.8\gamma_2 - \gamma_0$$

$$\begin{array}{ll} \gamma_1 \leq 4u^0 + v_1^0, & \gamma_1 \leq 4u^1 + v_1^1 \\ \gamma_2 \leq 5u^0 + v_2^0 + w^0, & \gamma_2 \leq 5u^1 + v_2^1 - w^1 \\ \gamma_3 \leq 3u^0 + v_3^0, & \gamma_3 \leq 3u^1 + v_3^1 \\ \gamma_4 \leq 6u^0 + v_4^0, & \gamma_4 \leq 6u^1 + v_4^1 \\ \gamma_0 \geq 8u^0 + v_1^0 + v_2^0 + v_3^0 + v_4^0, & \gamma_0 \geq 8u^1 + v_1^1 + v_2^1 + v_3^1 + v_4^1 - w^1 \\ u^0, u^1, v^0, v^1, w^0, w^1 \geq 0. \end{array}$$

Note that for the linear program to have a bounded optimal value, it is necessary to normalize the inequality. Some possibilities are $\sum_{j=1}^n \gamma_j \leq 1$, $\gamma_0 = 1$, $w^0 + w^1 = 1$, or $\sum_i (u_i^0 + v_i^0) + \sum_j (u_j^1 + v_j^1) + w^0 + w^1 = 1$. A 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. Several normalizations lead to the same inequality. \square

The lift-and-project linear program is approximately double the size of the description of P which means that the time to find a cut can be significant. It has been shown that a cut can be found using an LP that is close in size to the size of description of P . Take $P' = \{x : A'x \leq b'\}$ to be P possibly with the addition of some cuts, $x^* \in P'$ and (π, π_0) such that $\pi_0 < \pi x^* < \pi_0 + 1$.

The linear program is

$$\begin{aligned} \zeta &= \min(u^1 - u^2)b' + u^2 \frac{b' - A'x^*}{\pi x^*} \\ &\quad (u^1 - u^2)A' = \pi \\ &\quad u^1, u^2 \geq 0. \end{aligned}$$

Proposition 8.13 *Let $u = u^1 - u^2$. If $\zeta < \pi_0 + 1$, then*

$$u^1 \frac{b' - A'x}{ub} + u^2 \frac{b' - A'x}{1 - ub} \geq 1$$

is valid for $X = P' \cap \mathbb{Z}^n$ and cuts off x^ .*

For the knapsack instance of Example 8.16, this LP also produces the cut $1x_1 + \frac{1}{4}x_2 \leq 1$.

The idea of looking for the most violated inequality will be pursued in Chapter 9, in which we try to obtain “strong” inequalities.

8.9 Notes

- 8.2 The blossom inequality (4.3), derived again in Example 8.7, is from Edmonds [101].
- 8.3 The rounding procedure to generate cuts is from Gomory [153]. The general procedure described here and the proof of Theorem 8.1 for bounded integer programs is from Chvatal [70]. In Schrijver [265], the result is extended to unbounded polyhedra.
- 8.5 The first cutting plane proof reported is the procedure used to solve a 54-city TSP in Dantzig et al. [93].
- 8.6 The fractional cutting plane algorithm is presented in Gomory [153, 155]. 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 Gomory [154]. The presentation of mixed integer rounding inequalities is from Nemhauser and Wolsey [234]. For the theory of superadditive valid inequalities and superadditive duality, see Johnson [186]. Chapter II.1 in [233] presents a derivation of superadditive 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 Balas et al. [27] it is shown, more than 30 years after they were first proposed, that Gomory mixed integer cuts can be successfully used computationally. In Cook et al. [80], split cuts are introduced, and it is shown that if all possible split cuts are added, the result object, called the *split closure*, is again a polyhedron.
 Günlük and Pochet [170] present a new way to combine basic mixed integer inequalities leading to the so-called *mixing inequalities*.
- 8.8 Disjunctive and Gomory mixed integer cuts are closely related. Proposition 8.8 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 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 [21] in the 1970s, see also Jeroslow [185]. 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 been proposed to obtain tighter formulations for 0–1 MIPs by Lovasz and Schrijver [214], Balas et al. [25]

and Sherali and Adams [270]. The term *lift-and-project* was introduced in Balas et al. [25] and computational results are given in Balas et al. [26]. The reduced size LP treated in Proposition 8.13 that makes lift-and-project more tractable computationally is described in Bonami [56].

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) $X = \{x \in \{0,1\}^2 : 3x_1 - 4x_2 \leq 1\}$
 - (ii) $X = \{(x,y) \in \{0,1\} \times \mathbb{R}_+^1 : y \leq 20x, y \leq 7\}$
 - (iii) $X = \{(x,y) \in \mathbb{Z}^1 \times \mathbb{R}_+^1 : y \leq 6x, y \leq 16\}$.
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)

$$X = \{(x,y) \in \{0,1\} \times \mathbb{R}_+^2 : y_1 + y_2 \leq 2x, y_j \leq 1 \text{ for } j = 1, 2\}$$

$$(x, y_1, y_2) = (0.5, 1, 0)$$

(ii)

$$X = \{(x,y) \in \mathbb{Z}_+^1 \times \mathbb{R}_+^1 : y \leq 9, y \leq 4x\}$$

$$(x, y) = \left(\frac{9}{4}, 9\right)$$

(iii)

$$X = \{(x,y_1, y_2) \in \mathbb{Z}_+^1 \times \mathbb{R}_+^2 : y_1 + y_2 \leq 25, y_1 + y_2 \leq 8x\}$$

$$(x, y_1, y_2) = \left(\frac{25}{8}, 20, 5\right)$$

(iv)

$$X = \{x \in \mathbb{Z}_+^5 : 9x_1 + 12x_2 + 8x_3 + 17x_4 + 13x_5 \geq 50\}$$

$$x = \left(0, \frac{25}{6}, 0, 0, 0\right)$$

(v)

$$X = \{x \in \mathbb{Z}_+^4 : 4x_1 + 8x_2 + 7x_3 + 5x_4 \leq 33\}$$

$$x = \left(0, 0, \frac{33}{7}, 0\right).$$

3. Prove that $x_2 + x_3 + 2x_4 \leq 6$ is valid for

$$X = \{x \in \mathbb{Z}_+^4 : 4x_1 + 5x_2 + 9x_3 + 12x_4 \leq 34\}.$$

4. Consider the problem

$$\begin{aligned} \min \quad & x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 + x_2 \geq 4 \\ & x_1 + 5x_2 \geq 5 \\ & x \in \mathbb{Z}_+^2. \end{aligned}$$

- (i) Show that $x^* = \left(\frac{15}{4}, \frac{1}{4}\right)$ is the optimal linear programming solution.
 - (ii) Generate Gomory mixed integer cuts from the two rows of the optimal LP tableau taking the slack variables to be continuous variables.
 - (iii) Observing that the slack variables are integer, generate the Gomory fractional cuts and compare.
 - (iv) Solve the instance.
5. Solve $\min\{5x_1 + 9x_2 + 23x_3 : 20x_1 + 35x_2 + 95x_3 \geq 319, x \in \mathbb{Z}_+^3\}$ using Chvátal–Gomory inequalities or Gomory's cutting plane algorithm.
6. Solve $\max\{5x_1 + 9x_2 + 23x_3 - 4y : 2x_1 + 3x_2 + 9x_3 \leq 32 + y, x \in \mathbb{Z}_+^3, y \in \mathbb{R}_+^1\}$ using MIR inequalities.
7. (i) Show that the inequality $y_t \leq d_t x_t + s_t$ is valid for ULS.
(ii) Show that $y_t + y_{t+1} \leq (d_t + d_{t+1})x_t + d_{t+1}x_{t+1} + s_{t+1}$ is valid.
(iii) For $l \leq n, L = \{1, \dots, l\}$ and $S \subseteq L$, show that the inequality

$$\sum_{j \in S} y_j \leq \sum_{j \in S} \left(\sum_{t=j}^l d_t \right) x_j + s_l$$

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. Let $x_j = 1$ if j lies in the stable set. 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

$$(x_1 + 6x_2)/4 + x_3 + 4x_4 \geq 16$$

is a valid inequality for

$$X = \{x \in \mathbb{Z}_+^4 : x_1 + 6x_2 + 12x_3 + 48x_4 \geq 184\}.$$

- 10.** Use the mixed integer rounding procedure to show that

$$y_2 + y_4 \leq 20 + 4(x - 2)$$

is a valid inequality for $X =$

$$\{(x, y) \in \mathbb{Z}_+^1 \times \mathbb{R}_+^4 : y_1 + y_2 + y_3 + y_4 \leq 10x, y_1 \leq 13, y_2 \leq 15, y_3 \leq 6, y_4 \leq 9\}.$$

- 11.** (i) Show that if $\gamma x \leq \gamma_0 + \alpha(x_j - k)$ and $\pi x \leq \pi_0 + \beta(k + 1 - x_j)$ with $\alpha, \beta > 0$ and $k \in \mathbb{Z}^1$ are both valid for a polyhedron P , then $\gamma x \leq \gamma_0$ is valid for $P \cap \{x : x_j \in \mathbb{Z}^1\}$. An inequality generated in this way is a special split cut called a *D-inequality*.
(ii) * Show that if $P \subseteq \mathbb{R}^n$ is a polyhedron and $j \in \{1, \dots, n\}$, every valid inequality for $\text{conv}(P \cap \{x : x_j \in \{0, 1\}\})$ is or is dominated by a *D-inequality*.

- 12.** Prove Proposition 8.10.

- 13.** Consider an instance of the generalized transportation problem

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

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.

- 14.** Consider a telecommunications problem in which 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 six 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 $X = \{x \in \{0, 1\}^4 : x_i + x_j \leq 1 \text{ for all } 1 \leq i < j \leq 4\}$. defining stable sets. 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.

- 16.** Intersection Cuts.

Given the IP $\max\{cx : x \in P \cap \mathbb{Z}^n\}$, find an optimal LP basis as in (8.7). Now, dropping the nonnegativity on the basic variables and the integrality of the nonbasic variables (NB) gives the relaxation

$$x_B = \bar{a}_0 + \sum_{j \in \text{NB}} r^j x_j, \quad x_B \in \mathbb{Z}^m, x_N \in \mathbb{R}_+^{n-m},$$

where $r^j = -\bar{a}_j$ for $j \in \text{NB}$. Let $C \in \mathbb{R}^m$ be a convex set with $\bar{a}_0 \in \text{int}(C)$ and $\text{int}(C) \cap \mathbb{Z}^m = \emptyset$. Let $\alpha_j = \max\{\alpha : \bar{a}_0 + \alpha r^j \in C\}$. Show that $\sum_{j \in \text{NB}} \frac{x_j}{\alpha_j} \geq 1$ is a valid inequality for X . (Note that a split set $\pi_0 \leq \pi x \leq \pi_0 + 1$ is a special case of such a convex set.)

- 17.** An Extended Formulation.

Given the 0-1 integer program with $X = \{x \in \{0, 1\}^n : Ax \leq b\}$. For simplicity, take $m = 1$.

- (i) Choose some variable x_k and show that the nonlinear inequalities

$$\sum_{j=1}^n a_j x_j x_k \leq b x_k \quad \text{and} \quad \sum_{j=1}^n a_j x_j (1 - x_k) \leq b(1 - x_k)$$

are valid for X .

- (ii) Show that $x_k^2 = x_k$ is valid for X .

- (iii) Introduce additional variables $y_{ik} = x_i x_k$ for all $i \neq k$. Show that Q^k is a valid relaxation for X :

$$\begin{aligned} \sum_{j \neq k} a_j y_{jk} + a_k x_k &\leq b x_k \\ \sum_j a_j - \sum_{j \neq k} a_j y_{jk} - a_k x_k &\leq b(1 - x_k) \\ y_{ik} &\leq x_k \quad \text{for } i \neq k \\ y_{ik} &\leq x_i \quad \text{for } i \neq k \\ y_{ik} &\geq x_i + x_k - 1 \quad \text{for } i \neq k \\ x &\in [0, 1]^n, y_{ik} \in \mathbb{R}_+^{n-1}. \end{aligned}$$

- (iv) Observe a certain resemblance to the formulation in Proposition 8.10.
 (v) Repeat the reformulation for all variables x_j and set $y_{ij} = y_{ji}$ for all $i \neq j$.
 (vi) Repeat with pairs of variables, multiplying by $x_j x_k, x_j(1 - x_k), (1 - x_j)x_k$, and $(1 - x_j)(1 - x_k)$ and introducing variables y_{ijk} , etc.

9

Strong Valid Inequalities

9.1 Introduction

In the previous 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 disjunctive cuts, are 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,

- (i) 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.
- (ii) Describing interesting families \mathcal{F} of strong inequalities may be far from easy.
- (iii) 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 \mathcal{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 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 $\max\{cx : x \in X\}$, 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 $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$. 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.

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 $P \subseteq \mathbb{R}_+^n$, $\pi x \leq \pi_0$ *dominates* $\mu x \leq \mu_0$ on \mathbb{R}_+^n 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 $\mu x \leq \mu_0$ on \mathbb{R}_+^n , then $\{x \in \mathbb{R}_+^n : \pi x \leq \pi_0\} \subseteq \{x \in \mathbb{R}_+^n : \mu x \leq \mu_0\}$.

Definition 9.2 A valid inequality $\pi x \leq \pi_0$ is *redundant* in the description of $P \subset \mathbb{R}_+^n$, if there exist $k \geq 1$ valid inequalities $\pi^i x \leq \pi_0^i$ for $i = 1, \dots, k$ for P , and weights

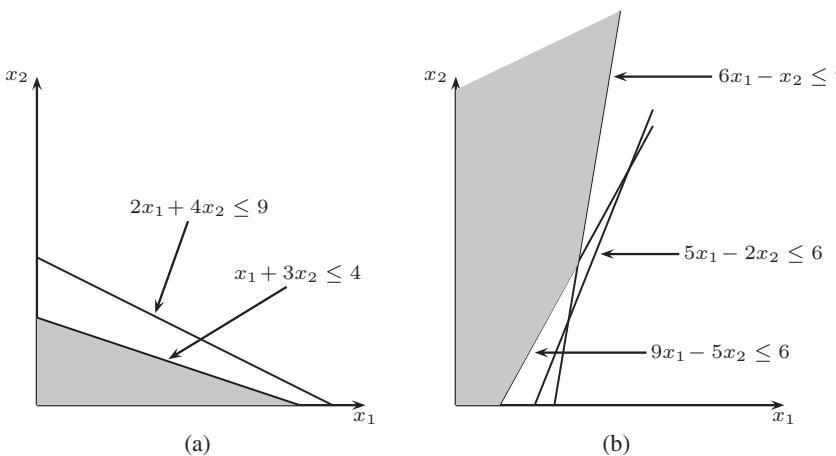


Figure 9.1 Two examples of dominated inequalities.

$u_i > 0$ for $i = 1, \dots, k$ such that $\left(\sum_{i=1}^k u_i \pi^i \right) x \leq \sum_{i=1}^k u_i \pi_0^i$ dominates $\pi x \leq \pi_0$ on \mathbb{R}_+^n .

Here we observe that $\{x \in \mathbb{R}_+^n : \pi^i x \leq \pi_0^i \text{ for } i = 1, \dots, k\} \subseteq \{x \in \mathbb{R}_+^n : (\sum_{i=1}^k u_i \pi^i) x \leq \sum_{i=1}^k u_i \pi_0^i\} \subseteq \{x \in \mathbb{R}_+^n : \pi x \leq \pi_0\}$.

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 $2x_1 + 4x_2 \leq 9$ on \mathbb{R}_+^2 . See Figure 9.1a.

Again with $n = 2$, suppose that $P = \{x \in \mathbb{R}_+^2 : 6x_1 - x_2 \leq 9, 9x_1 - 5x_2 \leq 6\}$. 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. \square

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.

Next we discuss polyhedra and characterize which inequalities are nonredundant.

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 $P \subseteq \mathbb{R}^n$ 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*

$$P = \{x \in \mathbb{R}^n : a^i x \leq b_i \text{ for } i = 1, \dots, m\},$$

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 $x^1, \dots, x^k \in \mathbb{R}^n$ 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 $(x^1, 1), \dots, (x^k, 1) \in \mathbb{R}^{n+1}$ 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 $P \subseteq \mathbb{R}^n$ is full-dimensional if and only if $\dim(P) = n$.

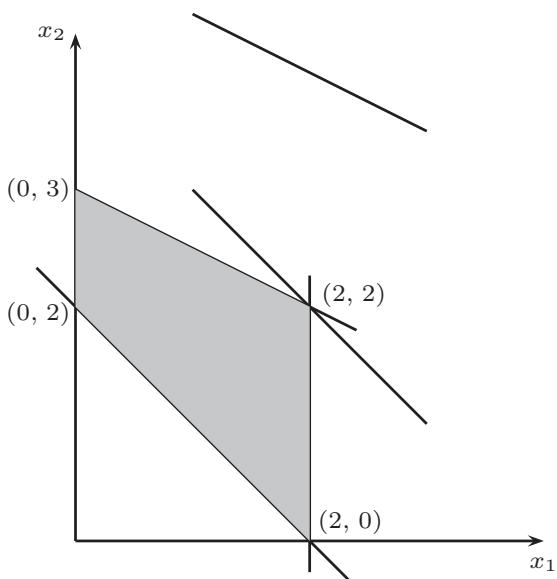
Definition 9.5

- (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 9.1 *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 .*

Figure 9.2 Facets and faces of a polyhedron.



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 $P \subset \mathbb{R}^2$, shown in Figure 9.2, described by the inequalities

$$\begin{array}{rcl} x_1 & \leq & 2 \\ x_1 + x_2 & \leq & 4 \\ x_1 + 2x_2 & \leq & 10 \\ x_1 + 2x_2 & \leq & 6 \\ x_1 + x_2 & \geq & 2 \\ x_1 & \geq & 0 \\ x_2 & \geq & 0. \end{array}$$

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, combining 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

$$\begin{array}{rcl} x_1 & \leq & 2 \\ x_1 + 2x_2 & \leq & 6 \\ x_1 + x_2 & \geq & 2 \\ x_1 & \geq & 0. \end{array}$$

□

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 $X \subset \mathbb{Z}_+^n$.

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 $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^m : \sum_{i=1}^m y_i \leq mx, y_i \leq 1 \text{ for } i = 1, \dots, m\}$ that arises in Sections 1.6 and 8.4 in formulating the uncapacitated facility location problem.

Problem 1. Given $X \subset \mathbb{Z}_+^n$ 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

$$\sum_{j=1}^n \mu_j x_j^k = \mu_0 \quad \text{for } k = 1, \dots, t$$

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 $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^m : \sum_{k=1}^m y_k \leq mx, y_i \leq 1 \text{ for } i = 1, \dots, m\}$, we have that $\dim(\text{conv}(X)) = m + 1$. Now we consider the valid inequality $y_i \leq x$ and show that it is facet-defining using Approach 2.

We select the simplest points $(0, \mathbf{0})$, $(1, e_i)$, and $(1, e_i + e_j)$ for $j \neq i$ that are feasible and satisfy $y_i = x$.

As $(0, \mathbf{0})$ lies on $\sum_{k=1}^m \mu_k y_k + \mu_{m+1} x = \mu_0$, $\mu_0 = 0$.

As $(1, e_i)$ lies on the hyperplane $\sum_{k=1}^m \mu_k y_k + \mu_{m+1} x = 0$, $\mu_i = -\mu_{m+1}$.

As $(1, e_i + e_j)$ lies on the hyperplane $\sum_{k=1}^m \mu_k y_k - \mu_i x = 0$, $\mu_j = 0$ for $j \neq i$.

So the hyperplane is $\mu_i y_i - \mu_i x = 0$, and $y_i \leq x$ is facet-defining. \square

Problem 2. Show that the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ 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 $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^m : \sum_{i=1}^m y_i \leq mx, y_i \leq 1 \text{ for } i = 1, \dots, m\}$, and consider the polyhedron/formulation

$$P = \{(x, y) \in \mathbb{R}^1 \times \mathbb{R}_+^m : y_i \leq x \text{ for } i = 1, \dots, m, x \leq 1\}.$$

Observe that the constraints $y_i - x \leq 0$ 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. So $P = \text{conv}(X)$. \square

Approach 2. Show that points $(x, y) \in P$ with x fractional are not extreme points of P .

Example 9.4 (cont.) Suppose that $(x^*, y^*) \in P$ with $0 < x^* < 1$. Note first that $(0, 0) \in P$. Also as $y_i^* \leq x^*$, the point $(\frac{y_1^*}{x^*}, \dots, \frac{y_m^*}{x^*}, 1) \in P$. But now

$$(x^*, y^*) = (1 - x^*)(0, \mathbf{0}) + x^* \left(1, \frac{y_1^*}{x^*}, \dots, \frac{y_m^*}{x^*}\right)$$

is a convex combination of two points of P and is not extreme. Thus, all vertices of P have x^* integer. \square

Approach 3. Show that for all $c \in \mathbb{R}^n$, 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 $Z^{LP} = \max\{fx + \sum_{i=1}^m c_i y_i : 0 \leq y_i \leq x \text{ for } i = 1, \dots, m, x \leq 1\}$. Consider an optimal solution (x^*, y^*) . Because

of the constraints $0 \leq y_i \leq x$, any optimal solution has $y_i^* = x^*$ if $c_i > 0$ and $y_i^* = 0$ if $c_i < 0$. The corresponding solution value is $(\sum_{i:c_i>0} c_i + f)x^*$ if $x^* > 0$ and 0 otherwise. Obviously if $(\sum_{i:c_i>0} c_i + f) > 0$, the objective is maximized by setting $x^* = 1$, and otherwise $x^* = 0$ is optimal. Thus, there is always an optimal solution with x integer, and $Z^{LP} = (f + \sum_{i:c_i>0} c_i)^+$. \square

Approach 4. Show that for all $c \in \mathbb{R}^n$, 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

$$\begin{aligned} \min t \\ w_i \geq c_i \quad \text{for } i = 1, \dots, m \\ t - \sum_{i=1}^m w_i \geq f \\ t \geq 0, w_i \geq 0 \quad \text{for } i = 1, \dots, m. \end{aligned}$$

Consider the two points $(0, \mathbf{0})$ and $(1, y^*)$ with $y_i^* = 1$ if $c_i > 0$ and $y_i^* = 0$ otherwise. Taking the better of the two leads to a primal solution of value $(f + \sum_{i:c_i>0} c_i)^+$. The point $w_i = c_i^+$ for $i = 1, \dots, m$ and $t = (f + \sum_{i:c_i>0} c_i)^+$ is clearly feasible in the dual. Thus, we have found a point in X and a dual solution of the same value. \square

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 $\sum_{i=1}^m \pi_i y_i + \pi_{m+1} x \leq \pi_0$. 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 $(0, \mathbf{0}) \in X$, $\pi_0 \geq 0$, and as $(1, e^S) \in X$, $\sum_{i \in S} \pi_i + \pi_{m+1} \leq \pi_0$, where e^S is the characteristic vector of S . Also a facet-defining inequality must have a tight point with $x = 1$. The point $(1, e^S)$ 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:

$$\begin{aligned} y_i - x \leq 0 \text{ with weight } \pi_i \text{ for } i \in S \\ -y_i \leq 0 \text{ with weight } -\pi_i \text{ for } i \in T \\ x \leq 1 \text{ with weight } \sum_{i \in S} \pi_i + \pi_{m+1}. \end{aligned}$$

The resulting inequality is $\sum_{i=1}^m \pi_i y_i + \pi_{m+1} x \leq \sum_{i \in S} \pi_i + \pi_{m+1}$. 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 . \square

Approach 6. Show that for any $c \in \mathbb{R}^n, c \neq \mathbf{0}$, 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: $\max f x + \sum_{i=1}^m c_i y_i$.

If $c_i > 0$ for some $i \in [1, m]$, $y_i = x$ in every optimal solution and so $M(f, c) \in \{(x, y) : y_i = x\}$.

If $c_i < 0$ for some $i \in [1, m]$, then $y_i = 0$ in every optimal solution.

If $c_i = 0$ for all i and $f > 0$, then $x = 1$ in every optimal solution.

If $c_i = 0$ for all i and $f < 0$, then $x = 0$ in any optimal solution.

All cases have been covered, and so $P = \text{conv}(X)$. \square

Approach 7. Verify that $b \in \mathbb{Z}^n$, and show that for all $c \in \mathbb{Z}^n$, the optimal value of the dual w^{LP} is integer valued. This is to show that the inequalities $Ax \leq b$ form a TDI system, see Theorem 3.8.

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. \square

Approach 8. (Projection from an Extended Formulation). Suppose $Q \subseteq \mathbb{R}^n \times \mathbb{R}^p$ is a polyhedron with $P = \text{proj}_x(Q)$ as defined in Section 1.7. Show that for all $c \in \mathbb{R}^n$, 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 setup variables x_1, \dots, x_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. \square

9.3 0–1 Knapsack Inequalities

Consider the set $X = \left\{ x \in \{0, 1\}^n : \sum_{j=1}^n a_j x_j \leq b \right\}$. 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 9.2 *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, $\sum_{j=1}^n a_j x_j^R = \sum_{j \in R} a_j \geq \sum_{j \in C} a_j > b$ and so $x^R \notin X$. \square

Example 9.6 Consider the knapsack set

$$X = \{x \in \{0, 1\}^7 : 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19\}.$$

Some minimal cover inequalities for X are:

$$\begin{array}{rcl} x_1 & +x_2 & +x_3 \\ x_1 & +x_2 & +x_6 \\ x_1 & & +x_5 & +x_6 \\ x_3 & +x_4 & +x_5 & +x_6 \end{array} \begin{array}{l} \leq 2 \\ \leq 2 \\ \leq 2 \\ \leq 3. \end{array}$$

\square

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 9.3 *If C is a cover for X , the extended cover inequality*

$$\sum_{j \in E(C)} x_j \leq |C| - 1 \tag{9.1}$$

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 9.2.

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 $2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$. \square

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 $\{x \in \{0, 1\}^4 : 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19\}$.

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 $\{x \in \{0, 1\}^5 : 11x_1 + 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19\}$?

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

if and only if $\alpha_1 + x_3 + x_4 + x_5 + x_6 \leq 3$ is valid for all $x \in \{0, 1\}^4$ satisfying $6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19 - 11$, or equivalently

if and only if $\alpha_1 + \max\{x_3 + x_4 + x_5 + x_6 : 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 8, x \in \{0, 1\}^4\} \leq 3$, or equivalently

if and only if $\alpha_1 \leq 3 - \zeta$, where $\zeta = \max\{x_3 + x_4 + x_5 + x_6 : 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 8, x \in \{0, 1\}^4\}$.

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. \square

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 variable x_2 , $\zeta_2 = \max\{2x_1 + x_3 + x_4 + x_5 + x_6 : 11x_1 + 6x_3 + 5x_4 + 5x_5 + 4x_6 \leq 19 - 6 = 13, x \in \{0, 1\}^5\} = 2$, 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 $2x_1 + 1x_2 + 1x_3 + 1x_4 + 1x_5 + 1x_6 + 0x_7 \leq 3$. \square

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 $X' = \{x \in \{0, 1\}^{|C|} : \sum_{j \in C} a_j x_j \leq b\}$. 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 $\zeta = \min_{C \subseteq N} \left\{ \sum_{j \in C} (1 - x_j^*) : \sum_{j \in C} a_j > b \right\} < 1$?

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:

$$\text{Is } \zeta = \min \left\{ \sum_{j \in N} (1 - x_j^*) z_j : \sum_{j \in N} a_j z_j > b, z \in \{0, 1\}^n \right\} < 1?$$

Theorem 9.2

- (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

$$X = \{x \in \{0, 1\}^6 : 45x_1 + 46x_2 + 79x_3 + 54x_4 + 53x_5 + 125x_6 \leq 178\}.$$

and the fractional point $x^* = (0, 0, \frac{3}{4}, \frac{1}{2}, 1, 0)$.

The cover separation problem is:

$$\begin{aligned} \zeta = \min \quad & 1z_1 + 1z_2 + \frac{1}{4}z_3 + \frac{1}{2}z_4 + 0z_5 + 1z_6 \\ & 45z_1 + 46z_2 + 79z_3 + 54z_4 + 53z_5 + 125z_6 > 178 \\ & z \in \{0, 1\}^6. \end{aligned}$$

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:

$$X = \left\{ (x, y) \in \{0, 1\}^n \times \mathbb{R}_+^n : \sum_{j \in N_1} y_j \leq b + \sum_{j \in N_2} y_j, y_j \leq a_j x_j \text{ for } j \in N_1 \cup N_2 \right\}.$$

Note that when $N_2 = \emptyset$ and $y_j = a_j x_j$ 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).

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.

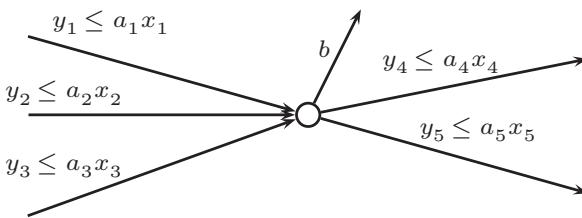


Figure 9.3 Single node flow set.

Proposition 9.4 *The flow cover inequality*

$$\sum_{j \in C_1} y_j + \sum_{j \in C_1} (a_j - \lambda)^+ (1 - x_j) \leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} x_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} y_j$$

is valid for X , where $L_2 \subseteq N_2 \setminus C_2$.

First to motivate the proof, suppose that $N_2 = \emptyset$. Let $lhs = \sum_{j \in C_1} y_j + \sum_{j \in C_1} (a_j - \lambda)^+ (1 - x_j)$ denote the left-hand side of the inequality and $rhs = b$ the right-hand side. Consider a feasible solution (x, y) . Let $R = \{j \in C_1 : a_j > \lambda, x_j = 0\}$. There are two possibilities.

If $R = \emptyset$, then $lhs = \sum_{j \in C_1} y_j \leq \sum_{j \in N_1} y_j \leq b = rhs$.

On the other hand, if $R \neq \emptyset$, then $y_j = 0$ for $j \in R$ and $|R| \geq 1$, so $lhs = \sum_{j \in C_1 \setminus R} y_j + \sum_{j \in R} (a_j - \lambda) \leq \sum_{j \in C_1} a_j - \lambda |R| \leq \sum_{j \in C_1} a_j - \lambda = b = rhs$.

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 $T = \{j \in N_1 \cup N_2 : x_j = 1\}$. There are two cases.

Case 1. $|C_1^+ \setminus T| + |L_2 \cap T| = 0$. Now

$$\begin{aligned} & \sum_{j \in C_1} y_j + \sum_{j \in C_1} (a_j - \lambda)^+ (1 - x_j) \\ &= \sum_{j \in C_1 \cap T} y_j + \sum_{j \in C_1^+ \setminus T} (a_j - \lambda) \\ &= \sum_{j \in C_1 \cap T} y_j \quad (\text{as } |C_1^+ \setminus T| = 0) \\ &\leq \sum_{j \in N_1} y_j \quad (\text{as } y_j \geq 0) \\ &\leq b + \sum_{j \in N_2} y_j \quad (\text{by definition of } X) \\ &= b + \sum_{j \in C_2} y_j + \sum_{j \in L_2 \cap T} y_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} y_j \end{aligned}$$

$$\begin{aligned} &\leq b + \sum_{j \in C_2} a_j + 0 + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} y_j \quad (\text{as } |L_2 \cap T| = 0) \\ &= b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} x_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} y_j. \end{aligned}$$

Case 2. $|C_1^+ \setminus T| + |L_2 \cap T| \geq 1$.

$$\begin{aligned} &\sum_{j \in C_1} y_j + \sum_{j \in C_1} (a_j - \lambda)^+ (1 - x_j) \\ &= \sum_{j \in C_1 \cap T} y_j + \sum_{j \in C_1^+ \setminus T} (a_j - \lambda) \\ &\leq \sum_{j \in C_1} a_j - |C_1^+ \setminus T| \lambda \quad (\text{as } y_j \leq a_j) \\ &\leq \sum_{j \in C_1} a_j - \lambda + \lambda |L_2 \cap T| \quad (\text{as } -|C_1^+ \setminus T| \leq -1 + |L_2 \cap T|) \\ &= b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} x_j \\ &\leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} x_j + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} y_j \quad (\text{as } y_j \geq 0). \end{aligned}$$

□

Example 9.8 Consider the set

$$X = \{(x, y) \in \{0, 1\}^6 \times \mathbb{R}_+^6 : y_1 + y_2 + y_3 \leq 4 + y_4 + y_5 + y_6,$$

$$y_1 \leq 3x_1, y_2 \leq 3x_2, y_3 \leq 6x_3, y_4 \leq 3x_4, y_5 \leq 5x_5, y_6 \leq 1x_6\}.$$

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

$$y_1 + y_3 + 1(1 - x_1) + 4(1 - x_3) \leq 7 + 2x_5 + y_6.$$

□

9.4.2 Separation for Flow Cover Inequalities*

Assuming that $y_j = a_j x_j$ 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:

$$\sum_{j \in C_1} a_j x_j + \sum_{j \in C_1} (a_j - \lambda)(1 - x_j) \leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in N_2 \setminus C_2} x_j,$$

or after simplification,

$$\sum_{j \in C_1} a_j - \lambda \sum_{j \in C_1} (1 - x_j) \leq b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in N_2 \setminus C_2} x_j.$$

Dividing by λ , this becomes

$$\sum_{j \in C_1} (1 - x_j) + \sum_{j \in N_2 \setminus C_2} x_j \geq 1,$$

or after rewriting,

$$\sum_{j \in C_1} (1 - x_j) - \sum_{j \in C_2} x_j \geq 1 - \sum_{j \in N_2} x_j.$$

This is nothing but the cover inequality for a 0–1 knapsack set with both positive and negative coefficients $\{x \in \{0, 1\}^n : \sum_{j \in N_1} a_j x_j - \sum_{j \in N_2} a_j x_j \leq b\}$ 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

$$\begin{aligned}\zeta &= \min \sum_{j \in N_1} (1 - x_j^*) z_j - \sum_{j \in N_2} x_j^* z_j \\ &\quad \sum_{j \in N_1} a_j z_j - \sum_{j \in N_2} a_j z_j > b \\ z &\in \{0, 1\}^n.\end{aligned}$$

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

$$\sum_{j \in C_1} y_j^* + \sum_{j \in C_1} (a_j - \lambda)^+ (1 - x_j^*) > b + \sum_{j \in C_2} a_j + \lambda \sum_{j \in L_2} x_j^* + \sum_{j \in N_2 \setminus (C_2 \cup L_2)} y_j^*$$

where $L_2 = \{j \in N_2 \setminus C_2 : \lambda x_j^* < y_j^*\}$. If so, a violated flow cover inequality has been found.

Example 9.8 (cont.) Suppose that a solution (x^*, y^*) is given with $y^* = (3, 0, 4, 3, 0, 0)$ and $x^* = (1, 0, \frac{2}{3}, 1, 0, 0)$. Solving the knapsack problem

$$\begin{aligned}\zeta &= \min 0 z_1 + 1 z_2 + \frac{1}{3} z_3 - 1 z_4 - 0 z_5 - 0 z_6 \\ &\quad 3 z_1 + 3 z_2 + 6 z_3 - 3 z_4 - 5 z_5 - z_6 > 4 \\ z &\in \{0, 1\}^6,\end{aligned}$$

one obtains as solution $C_1 = \{1, 3\}$, $C_2 = \{4\}$, and $\lambda = 2$. The resulting flow cover inequality

$$y_1 + y_3 + 1(1 - x_1) + 4(1 - x_3) \leq 7 + y_5 + y_6$$

is violated by $\frac{4}{3}$. □

As in the previous section, it is natural to ask if the flow cover inequality of Proposition 9.4 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 $y_j \leq a_j x_j$ 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 salesperson makes a profit f_j for visiting city (client) $j \in N$, pays travel costs c_e for traversing edge $e \in E$, but is not obliged to visit all the cities. The 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 $j \in N$, a possible formulation is:

$$\max - \sum_{e \in E} c_e x_e + \sum_{j \in N} f_j y_j \quad (9.2)$$

$$\sum_{e \in \delta(i)} x_e = 2y_i \quad \text{for } i \in N \quad (9.3)$$

$$\sum_{e \in E(S)} x_e \leq \sum_{i \in S \setminus \{k\}} y_i \quad \text{for } k \in S, S \subseteq N \setminus \{1\} \quad (9.4)$$

$$y_1 = 1 \quad (9.5)$$

$$x \in \{0, 1\}^{|E|}, y \in \{0, 1\}^{|N|}. \quad (9.6)$$

Constraints (9.3) ensure that if a node is visited, two adjacent edges are used to enter and leave, and (9.4) are *generalized subtour elimination* constraints (GSEC), which ensure that there is no subtour that does not contain node 1. Note that if we allow $x_e \in \{0, 1, 2\}$ for $e \in \delta(1)$, we also allow subtours 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

$$\zeta_k = \max \left\{ \sum_{e=(i,j) \in E' : i < j} x_e^* z_i z_j - \sum_{i \in N' \setminus \{k\}} y_i^* z_i : z \in \{0, 1\}^{|N'|}, z_k = 1 \right\}.$$

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

$$\zeta_k = \max \sum_{e \in E'} x_e^* w_e - \sum_{i \in N' \setminus \{k\}} y_i^* z_i \quad (9.7)$$

$$(IP) \quad w_e \leq z_i, w_e \leq z_j \quad \text{for } e = (i, j) \in E' \quad (9.8)$$

$$w_e \geq z_i + z_j - 1 \quad \text{for } e = (i, j) \in E' \quad (9.9)$$

$$w \in \{0, 1\}^{|E'|}, z \in \{0, 1\}^{|N'|}, z_k = 1. \quad (9.10)$$

Proposition 9.5 *If $x_e^* \geq 0$ for $e \in E'$, the linear program consisting of (9.7)–(9.9) and the bound constraints obtained by relaxing integrality in (9.10) 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 (9.8), $w_e = \min(z_i, z_j)$. As $z_i \leq 1$ for all $i \in N'$, $\min(z_i, z_j) \geq z_i + z_j - 1$ and so (9.9) 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 \square

The constraint matrix in the dual of the linear program in Proposition 9.5 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

$$n = 7, f = (2, 4, 1, 3, 7, 1, 7) \quad \text{and} \quad (c_e) = \begin{pmatrix} - & 4 & 3 & 3 & 5 & 2 & 5 \\ - & - & 5 & 3 & 3 & 4 & 7 \\ - & - & - & 4 & 6 & 0 & 4 \\ - & - & - & - & 4 & 4 & 6 \\ - & - & - & - & - & 5 & 8 \\ - & - & - & - & - & - & 3 \\ - & - & - & - & - & - & - \end{pmatrix}.$$

Iteration 0. We solve a linear programming relaxation of problem (9.2)–(9.6) consisting of the degree constraints (9.3), 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 $z^{LP} = 4$.

The separation problem for $k = 7$ is the linear program:

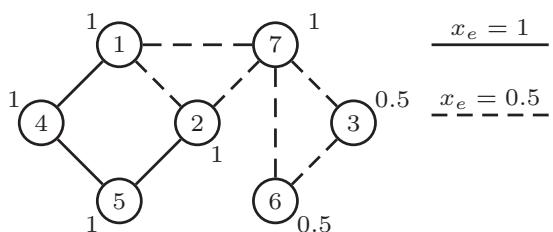
$$\zeta_7 = \max 1w_{45} + 1w_{25} + 0.5w_{27} + 0.5w_{67} + 0.5w_{37} + 0.5w_{36} \\ -1z_2 - 0.5z_3 - 1z_4 - 1z_5 - 0.5z_6$$

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 $z_3 = z_6 = z_7 = w_{36} = w_{37} = w_{67} = 1$ with violation $\xi_7 = \frac{1}{2}$.

Iteration 3. The constraint $x_{36} + x_{37} + x_{67} \leq y_3 + y_6$ is added.

Figure 9.4 Fractional optimal subtour solution.



The new linear programming solution is the single subtour $(1, 2, 5, 4, 7, 6)$, which is thus optimal with value of 2. \square

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 was also stressed and ideas for using the linear programming solution to try to construct feasible solutions are discussed in Chapter 13.

In this and Chapter 8, we have considered the development of valid inequalities along with separation algorithms so as to generate cuts that tighten up the formulations

A *Cut-and-Branch Algorithm* is a branch-and-bound algorithm in which cuts are generated at the top node of the tree before branch begins. 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, the introduction of cuts leads to a change in philosophy. Rather than reoptimizing fast at each node, the new viewpoint is that we are ready to work hard at a node if it enables us to obtain a tight dual bound at the node. So 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. There are also many new questions that arise concerning the use of cuts in the nodes, cut selection, cuts vs branching, locally valid vs globally valid cuts, etc.

A flowchart of the basic steps of a branch-and-cut algorithm is shown in Figure 9.5.

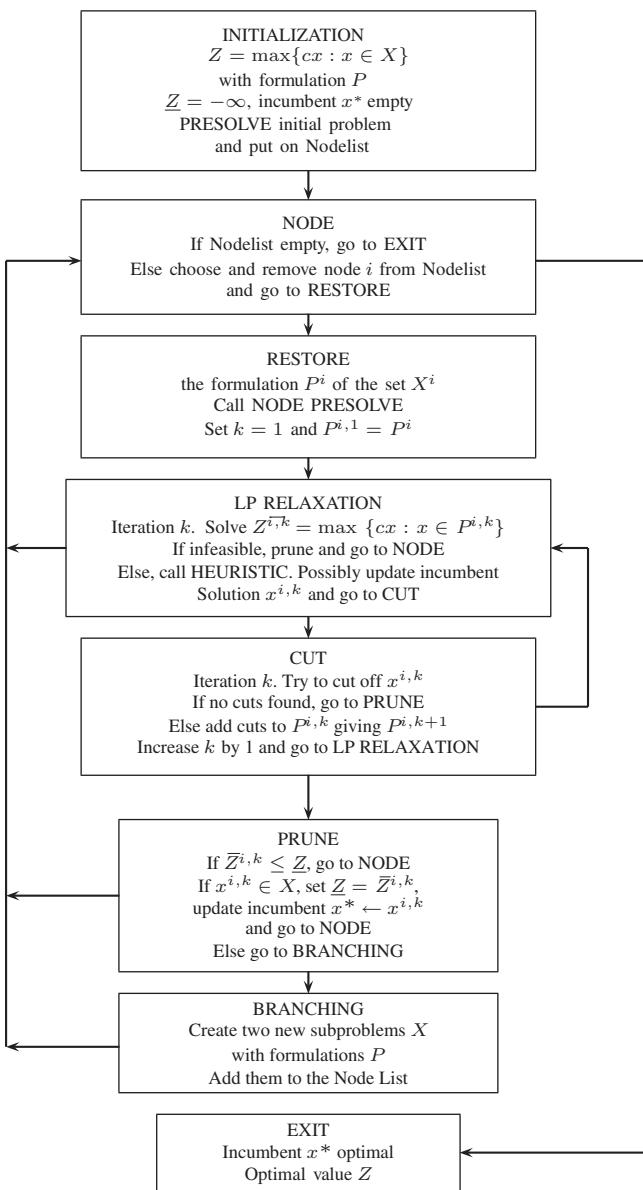


Figure 9.5 Branch-and-cut.

Example 9.10 0-1 Knapsack Problem. We consider the instance

$$\begin{aligned} \max \quad & 1x_1 + 8x_2 + 12x_3 + 16x_4 + 17x_5 + 22x_6 + 24x_7 \\ & 2x_1 + 4x_2 + 5x_3 + 13x_4 + 14x_5 + 15x_6 + 22x_7 \leq 31 \\ & x \in \{0, 1\}^7 \end{aligned}$$

We apply a simple branch-and-cut algorithm without preprocessing or heuristics that generates extended 0-1 cover inequalities (9.1) as cuts and is limited to three cuts at each node.

Node 1.

Iteration 1. $Z_{LP} = 50.615$, LP solution $(0, 1, 1, 0.538, 0, 1, 0)$

$C = \{3, 4, 6\}$ Inequality $x_3 + x_4 + x_6 + x_7 \leq 2$.

Iteration 2. $Z_{LP} = 50.5$, LP solution $(0, 1, 1, 0, 0.5, 1, 0)$

$C = \{2, 5, 6\}$ Inequality $x_2 + x_5 + x_6 + x_7 \leq 2$.

Iteration 3. $Z_{LP} = 48.417$, LP solution $(0, 1, 1, 0.583, 0.583, 0.417, 0)$

$C = \{3, 4, 5\}$ Inequality $x_3 + x_4 + x_5 + x_6 + x_7 \leq 2$.

3 cuts added, so decide to branch.

$Z_{LP} = 45.5$, LP solution $(0, 1, 0.125, 0.875, 0, 1, 0)$

Node 2. Branch $x_4 = 1$

Iteration 1. $Z_{LP} = 45$, LP solution $(0, 1, 0.1, 1, 0, 0.9, 0)$

$C = \{2, 4, 6\}$. Inequality $x_2 + x_4 + x_6 + x_7 \leq 2$.

Iteration 2. $Z_{LP} = 41$, LP solution $(0, 1, 0, 1, 1, 0, 0)$

New incumbent $\underline{Z} = 41$. Prune node by optimality.

Node 3. Branch $x_4 = 0$

Iteration 1. $Z_{LP} = 44.538$, LP solution $(1, 0.615, 0.615, 0, 0, 1, 0.385)$

$C = \{6, 7\}$ Inequality $x_6 + x_7 \leq 1$

Iteration 2. $Z_{LP} = 44.429$, LP solution $(1, 1, 1, 0, 0, 0.286, 0.714)$

$C = \{1, 2, 3, 7\}$ Inequality $x_1 + x_2 + x_3 + x_7 \leq 3$

Iteration 3. $Z_{LP} = 44$, LP solution $(0, 1, 1, 0, 0, 0, 1)$

New incumbent $\underline{Z} = 44$. Prune node by optimality.

STOP. Nodelist empty. Optimal solution. $Z = 44$, $x = (0, 1, 1, 0, 0, 0, 1)$

The corresponding branch-and-cut tree is shown in Figure 9.6. \square

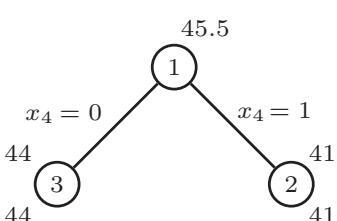


Figure 9.6 Branch-and-cut tree.

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, see Padberg [239], Nemhauser and Trotter [232] and the TSP polytope, see Grötschel and Padberg [165].

9.2 Books on polyhedra include Grunbaum [166], Stoer and Witzgall [271], and Ziegler [304]. Surveys specially written with integer programming and combinatorial optimization in view are Pulleyblank [253] and Chapter I.1.4 in [233].

9.3 Cover inequalities for 0–1 knapsack polytopes were developed simultaneously in Balas [22], Hammer et al. [172], and Wolsey [294]. The concept of lifting, already present in Gomory [157], was extended in Padberg [239] and Wolsey [295]. A major step forward came with the use of these inequalities in a cutting plane/branch-and-bound algorithm for 0–1 IPs in Crowder et al. [86]. Since around the year, 2000 separation routines for these inequalities were introduced into the commercial solvers such as CPLEX, GUROBI, and XPRESS. Since then different computational studies such as Bixby et al. [54], Achterberg and Wunderling [7] have shown that cuts are one of the most important elements in the codes.

There have also been many generalizations such as in Weismantel [289] for 0–1 knapsacks, and the generalization of cover inequalities to 0–1 knapsacks with GUB constraints in Wolsey [298] and to integer knapsacks in Ceria et al. [65].

9.4 Flow cover inequalities appeared first in Padberg et al. [242] and were implemented computationally in Van Roy and Wolsey [284]. Related inequalities for the model appear in Atamtürk [14]. Stronger liftings of the flow cover inequalities and extensive computational tests are presented in Gu et al. [167]. A mixed integer model consisting of a 0–1 knapsack constraint plus a single continuous variable is studied in Marchand and Wolsey [220]. Significant

generalizations can be found among others in Atamtürk and Günlük [15] and Atamtürk et al. [16].

- 9.5 The optimal subtour problem is studied in Balas [23]; see also Bauer [35] and more recently Jepsen et al. [184]. The GSEC inequalities are proposed by Goemans [149] 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 Rhys [258]. The problem of finding violated subtour elimination constraints quickly is faced by all the researchers developing computational studies of the TSP problem. Padberg and Rinaldi [241] present very effective heuristic separation routines. The major reference is now the book of Applegate et al. [12] devoted to the TSP problem.
- 9.6 An early survey on branch-and-cut is Jünger et al. [191]. See also the annotated bibliography of Caprara and Fischetti [63], 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. In the last 20 years, branch-and-cut has become the standard approach in the majority of computational studies involving discrete optimization.

Two publicly available research solvers designed so that the user can develop branch-and-cut applications are COIN [73] and SCIP [268]. However, the commercial codes such as CPLEX, XPRESS, and GUROBI also have subroutine libraries and call-backs allowing the user the possibility to develop specialized branch-and-cut algorithms.

9.8 Exercises

1. Consider the set $X = \{x \in Z_+^2 : x_1 - x_2 \geq -1, 2x_1 + 6x_2 \leq 15, x_1 - x_2 \leq 3, 2x_1 + 4x_2 \leq 7\}$. List and represent graphically the set of feasible points. Use this to find a minimal description of $\text{conv}(X)$.
2. Let $X = \{x \in \{0, 1\}^n : \sum_{j \in N} a_j x_j \leq b\}$ 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) $X = \{x \in \{0, 1\}^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$
 $x^* = \left(0, \frac{5}{8}, \frac{3}{4}, \frac{3}{4}, 0\right)$,

$$X = \{x \in \{0, 1\}^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$$

$$x^* = \left(0, \frac{5}{8}, \frac{3}{4}, \frac{3}{4}, 0\right),$$

(ii)

$$X = \{x \in \{0,1\}^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$$

$$x^* = \left(\frac{1}{4}, \frac{1}{8}, \frac{3}{4}, \frac{3}{4}, 0 \right),$$

(iii)

$$X = \{x \in \{0,1\}^5 : 7x_1 + 6x_2 + 6x_3 + 4x_4 + 3x_5 \leq 14\}$$

(iv)

$$x^* = \left(\frac{1}{7}, 1, \frac{1}{2}, \frac{1}{4}, 1 \right),$$

$$X = \{x \in \{0,1\}^5 : 12x_1 - 9x_2 + 8x_3 + 6x_4 - 3x_5 \leq 2\}$$

$$x^* = \left(0, 0, \frac{1}{2}, \frac{1}{6}, 1 \right).$$

4. Consider the knapsack set

$$X = \{x \in \{0,1\}^6 : 12x_1 + 9x_2 + 7x_3 + 5x_4 + 5x_5 + 3x_6 \leq 14\}.$$

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 $X = \{(x,y) \in \{0,1\} \times \mathbb{R}_+^4 : y_1 + y_2 + y_3 + y_4 \geq 36, y_1 \leq 20x_1, y_2 \leq 10x_2, y_3 \leq 10x_3, y_4 \leq 8x_4\}$
- (i) Derive a valid inequality that is a 0–1 knapsack constraint.
 - (ii) Use this to cut off the fractional point $y^* = (20, 10, 0, 6), x^* = (1, 1, 0, \frac{3}{4})$ with an inequality involving only x 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) $X = \{(x,y) \in \{0,1\}^3 \times \mathbb{R}_+^3 : y_1 + y_2 + y_3 \leq 7, y_1 \leq 3x_1, y_2 \leq 5x_2, y_3 \leq 6x_3\}$
 $(x^*, y^*) = \left(\frac{2}{3}, 1, 0; 2, 5, 0 \right)$.

- (ii) $X = \{(x,y) \in \{0,1\}^3 \times \mathbb{R}_+^3 : 7 \leq y_1 + y_2 + y_3, y_1 \leq 3x_1, y_2 \leq 5x_2, y_3 \leq 6x_3\}$
 $(x^*, y^*) = \left(\frac{2}{3}, 1, 0; 2, 5, 0 \right)$.

- (iii) $X = \{(x,y) \in \{0,1\}^6 \times \mathbb{R}_+^6 : y_1 + y_2 + y_3 \leq 4 + y_4 + y_5 + y_6, y_1 \leq 3x_1, y_2 \leq 3x_2, y_3 \leq 6x_3, y_4 \leq 3x_4, y_5 \leq 5x_5, y_6 \leq 1x_6\}$.
 $(x^*, y^*) = (1, 1, 0, 0, \frac{2}{5}, 0; 3, 3, 0, 0, 2, 0)$.

7. Consider the set $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^3 : y_1 + y_2 \leq b + y_3, y_2 \leq Mx\}$. Derive a flow cover inequality with $C = (\{2\}, \emptyset)$. For the constraints of ULS, $s_{t-1} + y_t = d_t + s_t, y_t \leq Mx_t, s_{t-1}, s_t \geq 0, x_t \in \{0, 1\}$, what is the resulting inequality?
8. Consider the set $X = \{(y_1, x_2, x_3, x_4) \in R_+^1 \times \{0, 1\}^3 : y_1 \leq 1000x_2 + 1500x_3 + 2200x_4, y_1 \leq 2000\}$. Show that it can be rewritten as a mixed 0–1 set

$$X' = \{(x, y) \in \{0, 1\}^4 \times \mathbb{R}_+^4 : y_1 \leq y_2 + y_3 + y_4,$$

$$y_1 \leq 2000x_1, y_2 \leq 1000x_2, y_3 \leq 1500x_3, y_4 \leq 2200x_4\}$$

with the additional constraints

$$x_1 = 1, y_2 \geq 1000x_2, y_3 \geq 1500x_3, y_4 \geq 2200x_4.$$

Use this to find a valid inequality for X cutting off

$$(y_1, x_2, x_3, x_4) = (2000, 0, 1, \frac{5}{22}).$$

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.
10. Solve the optimal subtour problem with $n = 6, f = (0, 19, 13, 1, 8, 12)$ and

$$(c_e) = \begin{pmatrix} - & 2 & 8 & 16 & 3 & 4 \\ - & 6 & 4 & 3 & 8 & \\ - & 10 & 12 & 9 & & \\ - & 5 & 7 & & & \\ - & 13 & & & & \\ - & & & & & \end{pmatrix}.$$

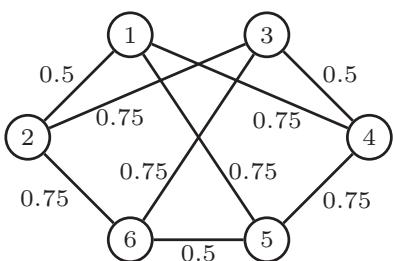


Figure 9.7 Fractional solution of STSP.

11. Solve the quadratic 0–1 problem

$$\max_{x \neq 0} \{x \in \{0, 1\}^5 : x_1 x_2 + 2x_1 x_3 + 3x_1 x_4 + 8x_1 x_5 + 1x_2 x_4 + 3x_2 x_5 + 6x_3 x_4 \\ + 5x_4 x_5 - x_1 - 4x_2 - 10x_3 - 2x_4 - 2x_5\}.$$

12. Consider the optimal subtree of a tree problem (Section 5.3) with an additional budget constraint, with feasible region: $X = \{x \in \{0, 1\}^n : x_j \leq x_{p(j)}$ 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 when $x_1 = 1$.

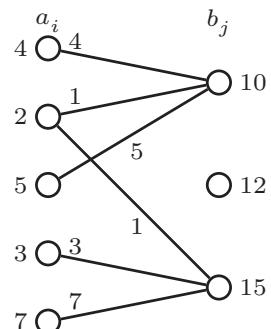
13. A possible formulation of the capacitated facility location problem is:

$$\sum_{j \in N} y_{ij} = a_i \quad \text{for } i \in M \\ \sum_{i \in M} y_{ij} \leq b_j x_j \quad \text{for } j \in N \\ y \in \mathbb{R}_+^{mn}, x \in \{0, 1\}^n.$$

Given an instance with $m = 5$ and $n = 3$, use the substitution $v_j = \sum_{i \in M} y_{ij}$ 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 y^* shown in Figure 9.8 and $x^* = (1, \frac{4}{15}, \frac{11}{15})$.

14. Given a graph $G = (V, E)$ with $n = |V|$, consider the set of incidence vectors of the stable sets $X = \{x \in \{0, 1\}^n : x_i + x_j \leq 1 \text{ for } e = (i, j) \in E\}$. Show that the clique inequality $\sum_{j \in C} x_j \leq 1$ 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 .

Figure 9.8 Fractional solution of CFL.



- 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 GAP with equality constraints

$$\begin{aligned} \max & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} = 1 \text{ for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \text{ for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Solve an instance with $m = 3, n = 2, (a_{ij}) = \begin{pmatrix} 5 & 7 \\ 3 & 8 \\ 2 & 10 \end{pmatrix}, (c_{ij}) = \begin{pmatrix} 20 & 16 \\ 15 & 19 \\ 19 & 14 \end{pmatrix}$,

and $b = \begin{pmatrix} 6 \\ 21 \end{pmatrix}$ with cutting planes.

- 17.** (*Superadditive Inequalities*) See Definition 2.6. Consider the feasible region $X = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$ and let $\mathcal{F} = \{F : \mathbb{R}^m \rightarrow \mathbb{R}^1 : F \text{ is superadditive, } F \text{ is nondecreasing, } F(0) = 0\}$.
- (i) Show that $\sum_{j=1}^n F(a_j)x_j \leq F(b)$ is a valid inequality for X .
 - (ii) Show that $F : \mathbb{R}^m \rightarrow \mathbb{R}^1$ with $F(d) = \lfloor \sum_{i=1}^n u_i d_i \rfloor$ is superadditive for $u \in \mathbb{R}_+^m$.
 - (iii) Show that if $F^1, F^2 \in \mathcal{F}$, then $F^* = \min(F^1, F^2) \in \mathcal{F}$.

10

Lagrangian Duality

10.1 Lagrangian Relaxation

Consider the integer program:

$$\begin{aligned} Z = \max cx \\ (\text{IP}) \quad Ax \leq b \\ \quad Dx \leq d \\ \quad x \in \mathbb{Z}_+^n. \end{aligned}$$

Suppose that the constraints $Dx \leq d$ are “nice” in the sense that an integer program with just these constraints (e.g. network constraints) is relatively easy. Thus, if we drop the “complicating constraints” $Ax \leq b$, the resulting relaxation is easier to solve than the original problem integer program (IP). Many problems have such a structure, for example, the traveling salesman problem if one drops the connectivity constraints, the uncapacitated facility location problem if one drops the client demand constraints, and so on. However, the resulting bound obtained from the relaxation may be weak because some important constraints are totally ignored. One way to tackle this difficulty is by *Lagrangian relaxation*, briefly introduced in Chapter 2.

We consider the problem IP in a slightly more general form:

$$\begin{aligned} Z = \max cx \\ Ax \leq b \\ x \in X \end{aligned}$$

where $Ax \leq b$ are m complicating constraints.

For any value of $u = (u_1, \dots, u_m) \geq 0$, we define the problem:

$$\begin{aligned} Z(u) = \max cx + u(b - Ax) \\ (\text{IP}(u)) \quad x \in X. \end{aligned}$$

Proposition 10.1 *Problem $\text{IP}(u)$ is a relaxation of problem IP for all $u \geq 0$.*

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.
Companion website: www.wiley.com/go/wolsey/integerprogramming2e

Proof. Remember that $\text{IP}(u)$ is a relaxation of IP if:

- (i) The feasible region is at least as large. This holds because $S = \{x : Ax \leq b, x \in X\} \subseteq X$.
- (ii) The objective value is at least as great in $\text{IP}(u)$ as in IP for all feasible solutions to IP . If $x \in S$ and $u \geq 0$, then $Ax \leq b$, $u(b - Ax) \geq 0$ and thus $cx + u(b - Ax) \geq cx$ for all $x \in S$. \square

We see that in $\text{IP}(u)$ the complicating constraints are handled by adding them to the objective function with a penalty term $u(b - Ax)$, or in other words, u is the *price* or *dual variable* or *Lagrange multiplier* associated with the constraints $Ax \leq b$.

Problem $\text{IP}(u)$ is called a *Lagrangian relaxation (subproblem) of IP with parameter u* . As $\text{IP}(u)$ is a relaxation of IP , $Z(u) \geq Z$ and we obtain an upper bound on the optimal value of IP . To find the best (smallest) upper bound over the infinity of possible values for u , we need to solve the *Lagrangian Dual Problem*:

$$(\text{LD}) \quad W_{\text{LD}} = \min\{Z(u) : u \geq 0\}.$$

Observation 10.1 When the m constraints that are dualized are equality constraints of the form $Ax = b$, the corresponding Lagrange multipliers $u \in \mathbb{R}^m$ are unrestricted in sign, and the Lagrangian dual (LD) becomes

$$W_{\text{LD}} = \min_u Z(u).$$

Solving the Lagrangian relaxation $\text{IP}(u)$ may sometimes lead to an optimal solution of the original problem IP .

Proposition 10.2 If $u \geq 0$,

- (i) $x(u)$ is an optimal solution of $\text{IP}(u)$, and
- (ii) $Ax(u) \leq b$, and
- (iii) $(Ax(u))_i = b_i$ whenever $u_i > 0$ (complementarity),
then $x(u)$ is optimal in IP .

Proof. By (i) $W_{\text{LD}} \leq Z(u) = cx(u) + u(b - Ax(u))$. By (iii) $cx(u) + u(b - Ax(u)) = cx(u)$. By (ii) $x(u)$ is feasible in IP and so $cx(u) \leq Z$. Thus, $W_{\text{LD}} \leq cx(u) + u(b - Ax(u)) = cx(u) \leq Z$. But as $W_{\text{LD}} \geq Z$, equality holds throughout and $x(u)$ is optimal in IP . \square

Note also that if the constraints dualized are equality constraints, condition (iii) is automatically satisfied. So an optimal solution to $\text{IP}(u)$ is optimal for IP if it is feasible in IP .

Consider now the application of this approach to the UFL problem, starting with the strong formulation:

$$\begin{aligned}
 Z = \max & \sum_{i \in M, j \in N} c_{ij} y_{ij} - \sum_{j \in N} f_j x_j \\
 (\text{IP}) \quad & \sum_{j \in N} y_{ij} = 1 \quad \text{for } i \in M \\
 & y_{ij} - x_j \leq 0 \quad \text{for } i \in M, j \in N \\
 & y \in \mathbb{R}_+^{|M| \times |N|}, \quad x \in \{0, 1\}^{|N|}.
 \end{aligned}$$

Dualizing the demand constraints gives:

$$\begin{aligned}
 Z(u) = \max & \sum_{i \in M, j \in N} (c_{ij} - u_i) y_{ij} - \sum_{j \in N} f_j x_j + \sum_{i \in M} u_i \\
 (\text{IP}(u)) \quad & y_{ij} \leq x_j \quad \text{for } i \in M, j \in N \\
 & y \in \mathbb{R}_+^{|M| \times |N|}, \quad x \in \{0, 1\}^{|N|}.
 \end{aligned}$$

This in turn breaks up into a subproblem for each location. Thus, $Z(u) = \sum_{j \in N} Z_j(u) + \sum_{i \in M} u_i$ where

$$\begin{aligned}
 Z_j(u) = \max & \sum_{i \in M} (c_{ij} - u_i) y_{ij} - f_j x_j \\
 (\text{IP}_j(u)) \quad & y_{ij} \leq x_j \quad \text{for } i \in M, \\
 & y_j \in \mathbb{R}_+^{|M|}, \quad x_j \in \{0, 1\}.
 \end{aligned}$$

Problem $\text{IP}_j(u)$ is easily solved by inspection. If $x_j = 0$, then $y_{ij} = 0$ for all i , and the objective value is 0. If $x_j = 1$, all clients that are profitable are served, namely those with $c_{ij} - u_i > 0$. The objective value is then $\sum_{i \in M} \max[c_{ij} - u_i, 0] - f_j$. So $Z_j(u) = \max\{0, \sum_{i \in M} \max[c_{ij} - u_i, 0] - f_j\}$.

Example 10.1 Uncapacitated facility location. Consider an instance with $m = 6$ clients and $n = 5$ potential locations, fixed location costs $f = (2, 4, 5, 3, 3)$ and the client-location profit matrix

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

Taking $u = (5, 6, 3, 2, 5, 4)$, we obtain the revised profit matrix

$$(c_{ij} - u_i) = \begin{pmatrix} 1 & -3 & -4 & -2 & 0 \\ -2 & 4 & -4 & 0 & -5 \\ 0 & -1 & 1 & -2 & 0 \\ 0 & -2 & 2 & -1 & 2 \\ -4 & 3 & 1 & -3 & 0 \\ -1 & -2 & 0 & 4 & -3 \end{pmatrix}$$

with $\sum_{i \in M} u_i = 25$. The resulting Lagrangian problem $IP(u)$ is then solved by inspection as described above. For instance, for $j = 2$, we obtain value 0 if $x_2 = 0$. On the other hand, if $x_2 = 1$ at a cost of 4, we can set $y_{22} = 1$ as $c_{22} - u_2 = 4 > 0$ and $y_{52} = 1$ as $c_{52} - u_5 = 3$, and thus the net profit with $x_2 = 1$ is $7 - 4 = 3$. Hence, it is optimal to set $x_2 = 1$ giving $Z_2(u) = 3$.

Carrying out a similar calculation for each depot, an optimal solution of $IP(u)$ is to set $x_1 = x_3 = x_5 = 0$, $x_2 = y_{22} = y_{52} = 1$, $x_4 = y_{64} = 1$ giving $Z(u) = 3 + 1 + \sum_{i \in M} u_i = 29$. \square

Now consider the application of Lagrangian relaxation to the symmetric traveling salesman problem (STSP). Remember that a *1-tree* is a tree on the graph induced by nodes $\{2, \dots, n\}$ plus two edges incident to node 1; see Section 2.3. So a tour is any 1-tree having two edges incident to each node.

Alternatively, we have seen that the problem of finding a minimum cost tour can be formulated as an integer program:

$$Z = \min \sum_{e \in E} c_e x_e \quad (10.1)$$

$$\sum_{e \in \delta(i)} x_e = 2 \quad \text{for } i \in V \quad (10.2)$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \text{for } 2 \leq |S| \leq |V| - 1 \quad (10.3)$$

$$x \in \{0, 1\}^{|E|}. \quad (10.4)$$

Observation 10.2 Half the subtour constraints (10.3) are redundant. For any feasible solution of the linear programming relaxation of this formulation, $|S| - \sum_{e \in E(S)} x_e = \frac{1}{2} \sum_{i \in S} \sum_{e \in \delta(i)} x_e - \sum_{e \in E(S)} x_e = \frac{1}{2} \sum_{e \in \delta(S, \bar{S})} x_e$ where $\delta(S, \bar{S})$ is the set of edges with one endpoint in S and the other in $\bar{S} = V \setminus S$. Therefore, as $\delta(S, \bar{S}) = \delta(\bar{S}, S)$, $|S| - \sum_{e \in E(S)} x_e = |\bar{S}| - \sum_{e \in E(\bar{S})} x_e$ and so $\sum_{e \in E(S)} x_e \leq |S| - 1$ if and only if $\sum_{e \in E(\bar{S})} x_e \leq |\bar{S}| - 1$. Note also that summing the constraints (10.2) and dividing by 2, we obtain $\sum_{e \in E} x_e = n$.

Therefore, we can drop all subtour elimination constraints with $1 \in S$. Now to obtain a Lagrangian relaxation, we dualize all the degree constraints on the nodes,

but also leave the degree constraint on node 1, and the constraint that the total number of edges is n , giving:

$$\begin{aligned} Z(u) &= \min \sum_{e \in E} (c_e - u_i - u_j)x_e + 2 \sum_{i \in V} u_i \\ &\quad \sum_{e \in \delta(1)} x_e = 2 \end{aligned} \tag{10.5}$$

$$(IP(u)) \quad \sum_{e \in E(S)} x_e \leq |S| - 1 \quad \text{for } 2 \leq |S| < |V|, 1 \notin S \tag{10.6}$$

$$\begin{aligned} \sum_{e \in E} x_e &= n \\ x &\in \{0, 1\}^{|E|}. \end{aligned} \tag{10.7}$$

The feasible solutions of $IP(u)$ are precisely the 1-trees. Finding a minimum weight 1-tree is easy, so this is potentially an interesting relaxation. Note also that as STSP is a minimization problem, its Lagrangian dual is a maximization problem. As the dualized constraints are equality constraints, the dual variables are unrestricted in sign. Thus here, $W_{LD} = \max_u Z(u)$.

Example 10.2 Consider an instance of STSP with edge cost matrix

$$(c_e) = \begin{pmatrix} - & 30 & 26 & 50 & 40 \\ - & - & 24 & 40 & 50 \\ - & - & - & 24 & 26 \\ - & - & - & - & 30 \\ - & - & - & - & - \end{pmatrix}.$$

Taking dual variables $u = (0, 0, -15, 0, 0)$ and writing $\bar{c}_e = c_e - u_i - u_j$, we obtain the revised cost matrix

$$(\bar{c}_e) = \begin{pmatrix} - & 30 & 41 & 50 & 40 \\ - & - & 39 & 40 & 50 \\ - & - & - & 39 & 41 \\ - & - & - & - & 30 \\ - & - & - & - & - \end{pmatrix}.$$

The optimal 1-tree is found by taking the two cheapest edges out of node 1, that is, edges (1, 2) and (1, 5), plus the edges of an optimal tree on nodes {2, 3, 4, 5}. Choosing greedily, the edges (4, 5), (2, 3), and (3, 4) are chosen in that order. $Z(u)$ is the sum of the length of this 1-tree and the value of $2 \sum u_i$, namely $178 - 30 = 148$. The resulting 1-tree is shown in Figure 10.1.

We have that $Z(u) = 148 \leq Z$. However, as the 1-tree is a tour, it is readily checked that its length is also 148. Thus, this tour is optimal. This also follows directly from Proposition 10.2. \square

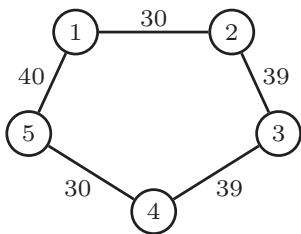


Figure 10.1 Optimal tour for STSP.

The questions that we would now like to answer are as follows:

- (i) How good is the upper bound obtained by solving the Lagrangian dual?
and
- (ii) How can one solve the Lagrangian dual?

10.2 The Strength of the Lagrangian Dual

To understand the Lagrangian dual problem (LD), we suppose for simplicity that the set X contains a very large but finite number of points $\{x^1, \dots, x^T\}$. Now

$$\begin{aligned} W_{\text{LD}} &= \min_{u \geq 0} Z(u) \\ &= \min_{u \geq 0} \{ \max_{x \in X} [cx + u(b - Ax)] \} \\ &= \min_{u \geq 0} \{ \max_{t=1, \dots, T} [cx^t + u(b - Ax^t)] \}. \end{aligned} \quad (10.8)$$

$$W_{\text{LD}} = \min \eta \quad (10.8)$$

$$\eta \geq cx^t + u(b - Ax^t) \quad \text{for all } t \quad (10.9)$$

$$u \in \mathbb{R}_+^m, \eta \in \mathbb{R}^1, \quad (10.10)$$

where the new variable η has been introduced to represent an upper bound on $Z(u)$. The latter problem is a linear program. Taking its dual gives

$$W_{\text{LD}} = \max \sum_{t=1}^T \mu_t(cx^t) \quad (10.11)$$

$$\sum_{t=1}^T \mu_t(Ax^t - b) \leq 0 \quad (10.12)$$

$$\sum_{t=1}^T \mu_t = 1 \quad (10.13)$$

$$\mu \in \mathbb{R}_+^T. \quad (10.14)$$

Now setting $x = \sum_{t=1}^T \mu_t x^t$ and using $\sum_{t=1}^T \mu_t = 1, \mu \in \mathbb{R}_+^T$, we get

$$W_{\text{LD}} = \max \{ cx : Ax - b \leq 0, x \in \text{conv}(X) \}$$

More generally it can be shown that the result still holds when $X = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ is the feasible region of any integer program.

Theorem 10.1 $W_{LD} = \max\{cx : Ax \leq b, x \in \text{conv}(X)\}$.

This theorem tells us precisely how strong a bound we obtain from dualization. In certain cases, it is no stronger than that of the linear programming relaxation.

Corollary 10.1 If $X = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ and $\text{conv}(X) = \{x \in \mathbb{R}_+^n : Dx \leq d\}$, then $W_{LD} = \max\{cx : Ax \leq b, Dx \leq d, x \in \mathbb{R}_+^n\}$.

For STSP, as constraints (10.5)–(10.7) with $x \geq 0$ describe the convex hull of the incidence vectors of 1-trees, this corollary tells us that W_{LD} gives us precisely the value of the linear programming relaxation of formulation (10.1)–(10.4). This is very interesting because it means that we have found a way to solve a linear program with an exponential number of constraints without treating them explicitly.

Definition 10.1 A function $f : \mathbb{R}^n \rightarrow \mathbb{R}^1$ is *convex* if for all $x^1, x^2 \in \mathbb{R}^n$ and $0 \leq \lambda \leq 1$, $f(\lambda x^1 + (1 - \lambda)x^2) \leq \lambda f(x^1) + (1 - \lambda)f(x^2)$. See Figure 10.2a.

The proof of Theorem 10.1 also tells us a good deal about the structure of the Lagrangian dual. It shows that the problem has been convexified so that it becomes a linear program. We have shown that

$$W_{LD} = \min_{u \geq 0} \{\max_{t=1,\dots,T} [cx^t + u(d - Dx^t)]\}.$$

Thus, the Lagrangian dual problem can also be viewed as the problem of minimizing the piecewise linear convex, but nondifferentiable function $Z(u)$ (see Figure 10.2b).

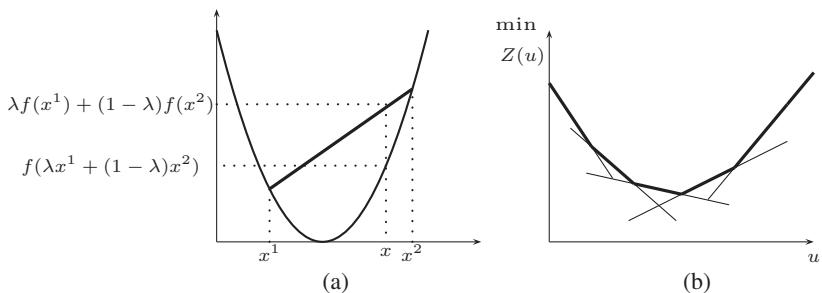


Figure 10.2 (a) A convex function. (b) Form of the dual problem.

10.3 Solving the Lagrangian Dual

The linear programming formulation (10.8)–(10.10) appearing in the proof of Theorem 10.1 provides one way to calculate W_{LD} , though the large number of constraints means that a constraint generation (or cutting plane) approach is required, as described in Chapter 8. Here, we describe a *subgradient algorithm* that is very simple and easy to implement without using a linear programming system.

The algorithm is designed to solve the problem of minimizing a piecewise linear convex function:

$$\min_{u \geq 0} f(u), \quad \text{where } f(u) = \max_{t=1,\dots,T} [a^t u - b_t].$$

In the case of the Lagrangian dual, we have

$$W_{LD} = \min_{u \geq 0} Z(u)$$

$$\text{where } Z(u) = \max_{t=1,\dots,T} [(b - Ax^t)u + cx^t].$$

A subgradient is a straightforward generalization of a gradient.

Definition 10.2 A *subgradient* at u of a convex function $f : \mathbb{R}^m \rightarrow \mathbb{R}^1$ is a vector $\gamma(u) \in \mathbb{R}^m$ such that $f(v) \geq f(u) + \gamma(u)^T(v - u)$ for all $v \in \mathbb{R}^m$.

For a continuously differentiable convex function f , $\gamma(u) = \nabla f(u) = \left(\frac{\partial f}{\partial u_1}, \dots, \frac{\partial f}{\partial u_m} \right)$ is the gradient of f at u .

Subgradient Algorithm for the Lagrangian Dual

Initialization. $u = u^0$.

Iteration k. $u = u^k$.

Solve the Lagrangian problem $IP(u^k)$ with optimal solution $x(u^k)$.

$$u_i^{k+1} = \max\{u_i^k - \mu_k(b - Ax(u^k)_i, 0) \mid i = 1, \dots, m\}$$

$$k \leftarrow k + 1$$

The vector $b - Ax(u^k)$ is easily shown to be a subgradient of $Z(u)$ at u^k . The simplicity of this algorithm is amazing. At each iteration, one takes a step from the present point u^k in the direction opposite to a subgradient. The difficulty is in choosing the *step lengths* $\{\mu_k\}_{k=1}^\infty$.

Theorem 10.2

(a) If $\sum_k \mu_k \rightarrow \infty$, and $\mu_k \rightarrow 0$ as $k \rightarrow \infty$, then $Z(u^k) \rightarrow W_{LD}$ the optimal value of LD.

- (b) If $\mu_k = \mu_0 \rho^k$ for some parameter $\rho < 1$, then $Z(u^k) \rightarrow W_{LD}$ if μ_0 and ρ are sufficiently large.
- (c) If $\bar{W} \geq W_{LD}$ and $\mu_k = \epsilon_k [Z(u^k) - \bar{W}] / ||b - Ax(u^k)||^2$ with $0 < \epsilon_k < 2$, then $Z(u^k) \rightarrow \bar{W}$, or the algorithm finds u^k with $\bar{W} \geq Z(u^k) \geq W_{LD}$ for some finite k .

This theorem tells us that rule (a) guarantees convergence, but as the series $\{\mu_k\}$ must be divergent (for example, $\mu_k = 1/k$), convergence is too slow to be of real practical interest.

On the other hand, step sizes (b) or (c) lead to much faster convergence, but each time with a possible inconvenience.

Using rule (b), the initial values of μ_0 and ρ must be sufficiently large, otherwise the geometric series $\mu_0 \rho^k$ tends to zero too rapidly, and the sequence u^k converges before reaching an optimal point. In practice, rather than decreasing μ_k at each iteration, a geometric decrease is achieved by halving the value of μ_k every v iterations, where v is some natural problem parameter such as the number of variables.

Using rule (c), the difficulty is that a dual upper bound $\bar{W} \geq W_{LD}$ is typically unknown. It is more likely in practice that a good primal lower bound $\underline{W} \leq W_{LD}$ is known. Such a lower bound \underline{W} is then used initially in place of \bar{W} . However, if $\underline{W} < W_{LD}$, the term $Z(u^k) - \underline{W}$ in the numerator of the expression for μ_k will not tend to zero, and so the sequences $\{u^k\}$, $\{Z(u^k)\}$ will not converge. If such behavior is observed, the value of \underline{W} must be increased.

For the STSP, care must be taken because here we are minimizing and the dualized constraints are the equality constraints

$$\sum_{e \in \delta(i)} x_e = 2 \quad \text{for all } i \in V.$$

The step direction is given by the rule

$$u_i^{k+1} = u_i^k + \mu_k \left(2 - \sum_{e \in \delta(i)} x_e(u^k) \right).$$

Here the i th coordinate of the subgradient $2 - \sum_{e \in \delta(i)} x_e(u^k)$ is two minus the number of arcs incident to node i in the optimal 1-tree. The step size, using rule (c), becomes

$$\mu_k = \epsilon_k (\underline{W} - Z(u^k)) / \sum_{i \in V} \left(2 - \sum_{e \in \delta(i)} x_e(u^k) \right)^2$$

where a lower bound \underline{W} on W_{LD} is appropriate because STSP is a minimization problem and LD a maximization problem.

Example 10.2 (cont.) Suppose that we have found the tour $(1, 2, 3, 4, 5, 1)$ of length 148 by some heuristic, but we have no lower bounds. We update the dual

variables using rule (c) from Theorem 10.2. We take $\epsilon_k = 1$ and, as no lower bound is available, we use the value $\bar{W} = 148$ of the tour.

Iteration 1. $u^1 = (0, 0, 0, 0, 0)$. The revised cost matrix is $\bar{c}^1 = c$. An optimal 1-tree is shown in Figure 10.3a, and $Z(u^1) = 130 \leq Z$. As $(2 - \sum_{e \in \delta(i)} x_e(u^k)) = (0, 0, -2, 1, 1)$, we have

$$u^2 = u^1 + [(148 - 130)/6](0, 0, -2, 1, 1).$$

Iteration 2. $u^2 = (0, 0, -6, 3, 3)$. The new cost matrix is

$$(\bar{c}_e^2) = \begin{pmatrix} - & 30 & 32 & 47 & 37 \\ - & - & 30 & 37 & 47 \\ - & - & - & 27 & 29 \\ - & - & - & - & 24 \\ - & - & - & - & - \end{pmatrix}.$$

We obtain $Z(u^2) = 143 + 2\sum_i u_i^2 = 143$, and

$$u^3 = u^2 + ((148 - 143)/2)(0, 0, -1, 0, 1).$$

The new optimal 1-tree is shown in Figure 10.3b.

Iteration 3. $u^3 = (0, 0, -17/2, 3, 11/2)$.

The new cost matrix is

$$(\bar{c}_e^3) = \begin{pmatrix} - & 30 & 34.5 & 47 & 34.5 \\ - & - & 32.5 & 37 & 44.5 \\ - & - & - & 29.5 & 29 \\ - & - & - & - & 21.5 \\ - & - & - & - & - \end{pmatrix}.$$

The new optimal 1-tree is shown in Figure 10.3c and we obtain the lower bound $Z(u^3) = 147.5$. As the cost data c are integral, we know that Z is integer valued and so $Z \geq \lceil 147.5 \rceil = 148$. As a solution of cost 148 is known, the corresponding solution has been proved optimal. \square

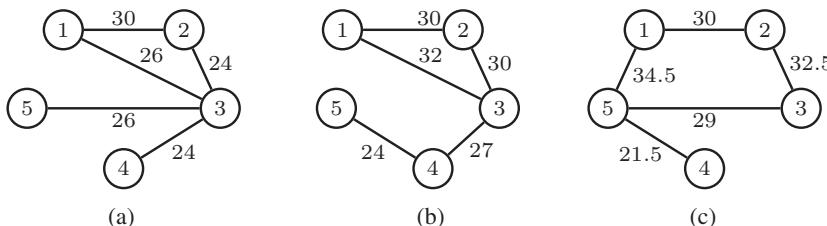


Figure 10.3 Optimal 1-tree for (a) \bar{c}^1 , (b) \bar{c}^2 , and (c) \bar{c}^3 .

The subgradient algorithm typically requires a large number of iterations. *Bundle methods* provide a way to reduce the number of iterations. Here the idea is (i) to bundle together some of the most recent subgradients so as to work with better information and (ii) to restrict the subproblem solutions $x(u)$ to remain not too far from a “central point” so as not to jump around erratically from one iteration to the next. The price to pay is the solution of a quadratic program rather than the simple update $u^{k+1} = \max\{u^k - \mu_k(b - Ax(u^k)), 0\}$ of the subgradient algorithm.

As the subgradient (and bundle) algorithms often terminate before the optimal value W_{LD} is attained and also as there is in most cases a duality gap ($W_{LD} > Z$), Lagrangian relaxation is typically embedded in a branch-and-bound algorithm. $W_{LD} > Z$ implies that the optimal solution of the primal linear program (10.11)–(10.14) has x nonintegral and this solution is typically what is used to make branching decisions in branch-and-bound. Unfortunately however, the subgradient algorithm does not provide a solution of this linear program. In addition, the solutions of the subproblem are integral, but almost always infeasible for the complicating constraints $Ax \leq b$. Some ideas on ways to overcome these difficulties are introduced in the section 10.4.

10.4 Lagrangian Heuristics

Here we consider simple ideas that have been used to convert the solution of the subgradient algorithm into feasible integer solutions and/or an approximate solution to the linear program (10.8)–(10.10) for use within a branch-and-bound algorithm.

Once the dual variables u begin to approach the set of optimal solutions, we hope to obtain a subproblem solution $x(u)$ that is “close” to being primal feasible every time that a Lagrangian subproblem IP(u) is solved. In the STSP, many nodes of the 1-tree will have degree 2, and so the solution is not far from being a tour, while for the UFL, many clients are served exactly once, and only a few are not served at all. Therefore, it is often straightforward to devise a simple heuristic that converts $x(u)$ into a feasible solution without greatly decreasing/increasing its value/cost. Below we examine this simple idea for set covering problems, as well as the possibility of fixing some variables once good primal and dual solutions are available.

Consider an instance of the set-covering problem

$$\min \left\{ \sum_{j \in N} c_j x_j : \sum_{j \in N} a_{ij} x_j \geq 1 \text{ for } i \in M, x \in \{0, 1\}^n \right\},$$

with $a_{ij} \in \{0, 1\}$ for $i \in M, j \in N$. The Lagrangian relaxation in which all the covering constraints are dualized is

$$Z(u) = \sum_{i \in M} u_i + \min \left\{ \sum_{j \in N} \left(c_j - \sum_{i \in M} u_i a_{ij} \right) x_j : x \in \{0, 1\}^n \right\}$$

for $u \geq 0$.

One simple possibility is to take an optimal solution $x(u)$ of this relaxation, drop all rows covered by the solution $x(u)$, that is, the rows $i \in M$ for which $\sum_{j \in N} a_{ij} x_j(u) \geq 1$, and solve the remaining smaller covering problem by a greedy heuristic. If x^G is the heuristic solution, then $x^H = x(u) + x^G$ is a feasible solution. It is then worth checking whether it cannot be improved by setting to zero some of the variables with $x_i(u) = 1$.

Once a heuristic solution has been found, it is also possible to use the Lagrangian for variable fixing. If \bar{Z} is the incumbent value, then any better feasible solution x satisfies $\sum_{i \in M} u_i + \min \sum_{j \in N} (c_j - \sum_{i \in M} u_i a_{ij}) x_j \leq cx < \bar{Z}$. Let $N_1 = \{j \in N : c_j - \sum_{i \in M} u_i a_{ij} > 0\}$ and $N_0 = \{j \in N : c_j - \sum_{i \in M} u_i a_{ij} < 0\}$.

Proposition 10.3 If $k \in N_1$ and $\sum_{i \in M} u_i + \sum_{j \in N_0} (c_j - \sum_{i \in M} u_i a_{ij}) + (c_k - \sum_{i \in M} u_i a_{ik}) \geq \bar{Z}$, then $x_k = 0$ in any better feasible solution.

If $x_k \in N_0$ and $\sum_{i \in M} u_i + \sum_{j \in N_0 \setminus \{k\}} (c_j - \sum_{i \in M} u_i a_{ij}) \geq \bar{Z}$, then $x_k = 1$ in any better feasible solution.

Example 10.3 Consider a set-covering instance with $m = 4$, $n = 6$,

$$c = (6, 7, 11, 5, 8, 5) \quad \text{and} \quad a_{ij} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

Taking $u = (4, 4, 3, 3)$, the Lagrangian subproblem $\text{IP}(u)$ takes the form

$$Z(u) = 14 + \min\{-1x_1 + 0x_2 + 1x_3 + 1x_4 + 1x_5 + 1x_6 : x \in \{0, 1\}^6\}.$$

An optimal solution is clearly $x(u) = (1, 0, 0, 0, 0, 0)$ with $Z(u) = 13$. The solution $x(u)$ covers rows 1 and 3, so the remaining problem to be solved heuristically is

$$\begin{array}{ccccccccc} \min & 7x_2 & + & 11x_3 & + & 5x_4 & + & 8x_5 & + & 5x_6 \\ & x_2 & & & & x_4 & + & x_5 & & \\ & & & x_3 & & & & x_5 & & \\ & & & x & \in & \{0,1\}^6 & & & & \end{array}$$

for which a greedy heuristic gives the solution $x^G = (0, 0, 0, 0, 1, 0)$. Adding together these two vectors, we obtain the heuristic solution $x^H = (1, 0, 0, 0, 1, 0)$ with cost 14. Thus, we now know that the optimal value lies between 13 and 14.

Now using Proposition 10.3, we see that with $N_0 = \{1\}$ and $N_1 = \{3, 4, 5, 6\}$, $x_1 = 1$ and $x_3 = x_4 = x_5 = x_6 = 0$ in any solution whose value is less than 14. However, fixing these values, the problem becomes infeasible. So the value of 14 is optimal. \square

Beyond just “correcting” an optimal dual solution $x(u)$ to obtain a primal feasible solution at small cost, a common idea is to count/weight the number of times a variable has been selected during the subgradient iterations. This information can be used to either approximate the “missing” optimal linear programming solution or to construct feasible integer solutions. A final step is to then to use this information to eliminate/fix a large number of variables and then solve a reduced size mixed integer program that may quickly provide improved feasible solutions.

10.5 Choosing a Lagrangian Dual

Suppose the problem to be solved is of the form:

$$\begin{aligned} Z &= \max c x \\ (\text{IP}) \quad A^1 x &\leq b^1 \\ A^2 x &\leq b^2 \\ x &\in \mathbb{Z}_+^n. \end{aligned}$$

If one wishes to tackle the problem by Lagrangian relaxation, there is a choice to be made. Should one dualize one or both sets of constraints, and if so, which set(s)? The answer must be based on a trade-off between

- (i) the *strength* of the resulting Lagrangian dual bound W_{LD} ,
- (ii) *ease of solution* of the Lagrangian subproblems $\text{IP}(u)$, and
- (iii) *ease of solution* of the Lagrangian dual problem: $W_{\text{LD}} = \min_{u \geq 0} Z(u)$.

Concerning (i), Theorem 10.1 gives us precise information about the strength of the bound.

Concerning (ii), the ease of solution of $\text{IP}(u)$ is problem specific. However, we know that if $\text{IP}(u)$ is “easy” in the sense of reducing to a linear program, that is $\text{IP}(u)$ involves maximization over $X = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ and $\text{conv}(X) = \{x \in \mathbb{R}_+^n : Dx \leq d\}$, then solving the linear programming relaxation of IP is an alternative to Lagrangian relaxation.

Concerning (iii), the difficulty using the subgradient (or other) algorithms is hard to estimate a priori, but the number of dual variables is at least some measure of the probable difficulty.

To demonstrate these trade-offs, consider the *Generalized Assignment Problem* (GAP):

$$\begin{aligned} Z = \max_x & \sum_{j=1}^n \sum_{i=1}^m c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} \leq 1 \quad \text{for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \quad \text{for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

We consider three possible Lagrangian relaxations. In the first, we dualize both sets of constraints giving $W_{\text{LD}}^1 = \min_{u \geq 0, v \geq 0} Z^1(u, v)$ where

$$\begin{aligned} Z^1(u, v) = \max_x & \sum_{j=1}^n \sum_{i=1}^m (c_{ij} - u_i - a_{ij} v_j) x_{ij} + \sum_{i=1}^m u_i + \sum_{j=1}^n v_j b_j \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Here we dualize the first set of assignment constraints giving $W_{\text{LD}}^2 = \min_{u \geq 0} Z^2(u)$ where

$$\begin{aligned} Z^2(u) = \max_x & \sum_{j=1}^n \sum_{i=1}^m (c_{ij} - u_i) x_{ij} + \sum_{i=1}^m u_i \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \quad \text{for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}, \end{aligned}$$

and here we dualize the knapsack constraints giving $W_{\text{LD}}^3 = \min_{v \geq 0} Z^3(v)$ where

$$\begin{aligned} Z^3(v) = \max_x & \sum_{j=1}^n \sum_{i=1}^m (c_{ij} - a_{ij} v_j) x_{ij} + \sum_{j=1}^n v_j b_j \\ & \sum_{j=1}^n x_{ij} \leq 1 \quad \text{for } i = 1, \dots, m \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Based on Theorem 10.1, we know that $W_{\text{LD}}^1 = W_{\text{LD}}^3 = Z_{\text{LP}}$ as for each i , $\text{conv}\{x : \sum_{j=1}^n x_{ij} \leq 1, x_{ij} \in \{0, 1\} \text{ for } j = 1, \dots, n\} = \{x : \sum_{j=1}^n x_{ij} \leq 1, 0 \leq x_{ij} \leq 1 \text{ for } j = 1, \dots, n\}$. The values of $Z^1(u, v)$ and $Z^3(v)$ can both be calculated by inspection. To calculate $Z^1(u, v)$, note that the problem decomposes variable by variable, while for $Z^3(v)$ the problem decomposes into a simple problem for each $j = 1, \dots, n$. In terms of solving the Lagrangian dual problems, calculating W_{LD}^3 appears easier than calculating W_{LD}^1 because there are only n as opposed to $m+n$ dual variables.

The second relaxation potentially gives a tighter bound $W_{\text{LD}}^2 \leq Z_{\text{LB}}$ as in general for fixed j , $\text{conv} \left\{ x : \sum_{i=1}^m a_{ij}x_{ij} \leq b_j, x_{ij} \in \{0, 1\}^m \right\} \subset \{x : \sum_{i=1}^m a_{ij}x_{ij} \leq b_j, 0 \leq x_{ij} \leq 1 \text{ for } i = 1, \dots, m\}$. However, here the Lagrangian subproblem involves the solution of n 0–1 knapsack problems.

10.6 Notes

- 10.1 Many of the properties of the Lagrangian dual can be found in Everett [108]. The successful solution of what were at the time very large TSPs by Held and Karp [174, 175] made the approach popular.
- 10.2 The application of Lagrangian relaxation to integer programming and in particular Theorem 10.1 and its consequences were explored in Geoffrion [137]. Early surveys of applications include Fisher [119] and Beasley [38].
- 10.3 Simple multiplier adjustment methods have also been tried for various problems, among them the uncapacitated facility location problem by Erlenkotter [107]. The use of the subgradient algorithm to solve the Lagrangian dual again stems from Held and Karp [174]. Early studies and analysis of the subgradient approach can be found in Held et al. [176] and Goffin [150]. Lemaréchal [205] is a survey on Lagrangian relaxation discussing the ties between row generation, subgradient, and bundle algorithms. Barahona and Anbil [32] present a volume algorithm, a modification of the subgradient algorithm so as to also obtain a primal LP solution. The effectiveness of modern subgradient methods to obtain Lagrangian bounds is studied in Frangioni et al. [126]. Briant et al. [62] compare bundle methods and column generation that is treated in Chapter 11.
- 10.4 Many problems have been tackled using Lagrangian heuristics, in particular a wide variety of location problems, many fixed cost network design problems and several production planning problems. Recent examples include Jena et al. [182] for a dynamic facility location problem, Mirchandani and Borenstein [229] on vehicle rescheduling, Eksioglu et al. [106] on integrated production and transport planning, Brahimi and Dauzère-Pérès [59] on a capacitated lot-sizing problem, and Cunha et al. [87] on quadratic knapsack problems. To obtain exact algorithms, the heuristics are then often integrated into column generation algorithms (see Chapter 11).
- 10.5 A comparison of different Lagrangian relaxations for the capacitated facility location problem can be found in Cornuéjols et al. [82]. By duplicating variables, dualizing the equations identifying variables, and solving separate subproblems for each set of distinct variables, Lagrangian relaxation can be used to get stronger bounds in certain cases; see Exercise 6. Lagrangian decomposition is one of several names given to this idea that appeared in Jornsten and Nasberg [189] and Guignard and Kim [168].

10.7 Exercises

1. Consider an instance of UFL with $m = 6$, $n = 5$, delivery costs

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

and fixed costs $f = (4, 8, 11, 7, 5)$. Using the dual vector $u = (5, 6, 3, 2, 6, 4)$, solve the Lagrangian subproblem $\text{IP}(u)$ to get an optimal solution $(x(u), y(u))$ and lower bound $Z(u)$. Modify the dual solution $(x(u), y(u))$ to construct a good primal feasible solution. How far is this solution from optimal?

2. Suppose one dualizes the constraints $y_{ij} \leq x_j$ in the strong formulation of UFL.
- (i) How strong is the resulting Lagrangian dual bound?
 - (ii) How easy is the solution of the Lagrangian subproblem?
3. Use Lagrangian relaxation to solve the STSP instance with distances

$$(c_e) = \begin{pmatrix} - & 8 & 2 & 14 & 26 & 13 \\ - & - & 7 & 4 & 16 & 8 \\ - & - & - & 23 & 14 & 9 \\ - & - & - & - & 12 & 6 \\ - & - & - & - & - & 5 \end{pmatrix}.$$

4. Consider GAP with equality constraints

$$\begin{aligned} \max & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} = 1 \quad \text{for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \quad \text{for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Solve an instance with $m = 3$, $n = 2$, $(a_{ij}) = \begin{pmatrix} 5 & 7 \\ 3 & 8 \\ 2 & 10 \end{pmatrix}$, $(c_{ij}) = \begin{pmatrix} 20 & 16 \\ 15 & 19 \\ 19 & 14 \end{pmatrix}$, and $b = \begin{pmatrix} 6 \\ 21 \end{pmatrix}$ with Lagrangian relaxation.

5. Consider a 0–1 knapsack problem

$$\begin{aligned} Z = \max \quad & 11x_1 + 5x_2 + 14x_3 \\ \text{s.t. } & 3x_1 + 2x_2 + 4x_3 \leq 5 \\ & x \in \{0, 1\}^3. \end{aligned}$$

Construct a Lagrangian dual by dualizing the knapsack constraint.

- (i) What is the optimal value of the dual variable?
 - (ii) Suppose one runs the subgradient algorithm using step size (b) in Theorem 10.2, starting with $u^0 = 2$, $\mu_0 = \frac{1}{2}$ and $\rho = \frac{1}{2}$. Show numerically that the subgradient algorithm does not reach the optimal dual solution.
 - (iii) Given that $L(u)$ is a 1-dimensional piecewise-linear convex function, suggest an alternative to the subgradient algorithm.
6. *Lagrangian Decomposition.* Consider the problem $Z = \max\{cx : A^i x \leq b^i \text{ for } i = 1, 2, x \in \mathbb{Z}_+^n\}$ with the reformulation
- $$\max\{\alpha cx^1 + (1 - \alpha)cx^2 : A^i x^i \leq b^i \text{ for } i = 1, 2, x^1 - x^2 = 0, x^i \in \mathbb{Z}_+^n \text{ for } i = 1, 2\}$$
- for $0 < \alpha < 1$.
- Consider the Lagrangian dual of this formulation in which the n constraints $x^1 - x^2 = 0$ are dualized. What is the strength of this dual?
7. Consider the capacitated facility location problem

$$\begin{aligned} \min \quad & \sum_{i \in M} \sum_{j \in N} c_{ij} y_{ij} + \sum_{j \in N} f_j x_j \\ \text{s.t. } & \sum_{j \in N} y_{ij} = a_i \quad \text{for } i \in M \\ & \sum_{i \in M} y_{ij} \leq b_j x_j \quad \text{for } j \in N \\ & \sum_{j \in N} y_{ij} \leq \min\{a_i, b_j\} x_j \quad \text{for } i \in M, j \in N \\ & \sum_{j \in N} b_j x_j \geq \sum_{i \in M} a_i \\ & y \in \mathbb{R}_+^{mn}, x \in \{0, 1\}^n. \end{aligned}$$

Discuss the advantages and disadvantages of different Lagrangian relaxations. Which would you choose to implement and why?

8. Consider the assignment problem with budget constraint

$$\begin{aligned} \max \quad & \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j \in N} x_{ij} = 1 \quad \text{for } i \in M \\ & \sum_{i \in M} x_{ij} = 1 \quad \text{for } j \in N \\ & \sum_{i \in M} \sum_{j \in N} a_{ij} x_{ij} \leq b \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Discuss the strength of different possible Lagrangian relaxations, and the ease or difficulty of solving the Lagrangian subproblems, and the Lagrangian dual.

9. *Surrogate Duality.* Given the IP: $Z = \max\{cx : Ax \leq b, x \in X\}$, consider the *surrogate relaxation* $Z_{\text{SR}}(u) = \max\{cx : uAx \leq ub, x \in X\}$ with $u \in \mathbb{R}_+^m$ and the *surrogate dual* $W_{\text{SD}} = \min_{u \in \mathbb{R}_+^m} Z_{\text{SR}}(u)$ and the Lagrangian dual $W_{\text{LD}} = \min_{u \in \mathbb{R}_+^m} \{(c - uA)x + ub : x \in X\}$.

Show that $W_{\text{SD}} \leq W_{\text{LD}}$.

11

Column (and Row) 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 $X = \cap_{k=0}^K X^k$ for some $K \geq 1$. Even more particular is the case where the constraints take the form:

$$\begin{array}{llllll}
 A^1x^1 & +A^2x^2 & +\cdots & +A^Kx^K & \leq b \\
 D^1x^1 & & & & \leq d_1 \\
 (\text{IP}) & \dots & & & \leq . \\
 & & & & \leq . \\
 & & & & \dots & \leq . \\
 & & & & D^Kx^K & \leq d_K \\
 x^1 \in Z_+^{n_1}, & \dots & \dots & , x^K \in Z_+^{n_K}, & &
 \end{array}$$

so that the sets $X^k = \{x^k \in Z_+^{n_k} : D^kx^k \leq d_k\}$ are independent for $k = 1, \dots, K$, and only the *joint* constraints $\sum_{k=1}^K A^kx^k \leq b$ link together the different sets of variables. The generalized assignment problem discussed in Chapter 10 is of this form when we take the assignment constraints to be the complicating constraints and the separate knapsack sets provide the sets X^k for $k = 1, \dots, n$. On the other hand, consider

The Resource-Constrained Shortest Path Problem. Given a directed graph $D = (V, A)$ with $s, t \in V$, costs $c \in \mathbb{R}_+^{|A|}$ and traversal times $\tau \in \mathbb{R}_+^{|A|}$ for the arcs, the problem is to find a minimum cost $s-t$ path whose traversal time does not exceed b . Letting $x_{ij} = 1$ if arc (i, j) is in the chosen path, an IP formulation is

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.

Companion website: www.wiley.com/go/wolsey/integerprogramming2e

$$\begin{aligned}
& \min \sum_{(i,j) \in A} c_{ij} x_{ij} \\
& \sum_{(s,j) \in \delta^+(s)} x_{sj} = 1 \\
& \sum_{(i,j) \in \delta^-(i)} x_{ij} - \sum_{(i,j) \in \delta^+(i)} x_{ij} = 0 \quad \text{for } i \in V \setminus \{s, t\} \\
& \sum_{(i,j) \in A} \tau_{ij} x_{ij} \leq b \\
& x \in \{0, 1\}^{|A|}
\end{aligned}$$

where $\delta^+(i) = \{(i, j) \in A\}$ and $\delta^-(i) = \{(j, i) \in A\}$. Taking the first two sets of constraints defining an s - t path as the complicating constraints and the third constraint limiting the travel time as the subproblem, here one has just a single $K = 1$ subproblem.

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 the calculation

$$L(u) = \max \left\{ \sum_{k=1}^K (c^k - u A^k) x^k + ub : x^k \in X^k \text{ for } k = 1, \dots, K \right\}$$

breaks up into K distinct subproblems.

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 extreme points of $\text{conv}(X^k)$ (or each of the feasible points in X^k) and each λ^k is a vector of corresponding variables.

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, arise by reformulation. We then examine how to solve the linear programming relaxation of these Master Problems using the solution of the *Column Generation Subproblem(s)*. We also relate the strength of this relaxation to that obtained by Lagrangian duality, or by the use of cutting planes. Next 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. Finally, we also allow the addition of cutting planes giving rise to *Branch-Cut-and-Price* algorithms.

11.2 The Dantzig–Wolfe Reformulation of an IP

For simplicity in describing the approach, we suppose that there is just one subproblem $K = 1$. Later, we discuss the minor changes required when $K > 1$. We consider the problem

$$Z_{\text{IP}} = \max\{cx : Ax \leq b, x \in X\}, \quad (11.1)$$

where $X = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ and $S = X \cap \{x : Ax \leq b\}$ is the complete set of feasible solutions. As for Lagrangian relaxation, we assume that there is an algorithm available allowing us to optimize over X repeatedly and we view $Ax \leq b$ as complicating constraints. An equivalent problem is:

$$Z_{\text{IP}} = \max\{cx : Ax \leq b, x \in \text{conv}(X), x \in \mathbb{Z}_+^n\}. \quad (11.2)$$

Now as $\text{conv}(X) = \left\{x \in \mathbb{R}^n : x = \sum_{t=1}^T z^t \lambda_t, \sum_{t=1}^T \lambda_t = 1, \lambda \in \mathbb{R}_+^T\right\}$ where $\{z^t\}_{t=1}^T$ are the extreme points of $\text{conv}(X)$, we obtain, after substituting for x , the equivalent Master Problem:

$$Z_M = \max \sum_{t=1}^T (cz^t) \lambda_t \quad (11.3)$$

$$\sum_{t=1}^T (Az^t) \lambda_t \leq b \quad (11.4)$$

$$(M) \quad \sum_{t=1}^T \lambda_t = 1 \quad (11.5)$$

$$\lambda \in \mathbb{R}_+^T \quad (11.6)$$

$$x - \sum_{t=1}^T z^t \lambda_t = 0 \quad (11.7)$$

$$x \in \mathbb{Z}^n. \quad (11.8)$$

The above substitution is referred to as *convexification*. The alternative, called *discretization*, is to take $\{z^t\}_{t=1}^T$ to be all the points of X . In this case, it is possible to drop the constraints (11.7) and (11.8) on x and replace (11.6) by $\lambda \in \{0, 1\}^T$. Note that when $X \subseteq \{0, 1\}^n$, the two approaches coincide because all the points of X are extreme points of $\text{conv}(X)$.

11.3 Solving the LP Master Problem: Column Generation

Here we consider how to solve the linear programming relaxation of the Master Problem. Whether we are using the extreme points of $\text{conv}(X)$ or all the points of X , we obtain the same linear programming relaxation:

$$(LM) \quad \begin{aligned} Z_{\text{LM}} &= \max \sum_{t=1}^T (cz^t)\lambda_t \\ \sum_{t=1}^T (Az^t)\lambda_t &\leq b \\ \sum_{t=1}^T \lambda_t &= 1 \\ \lambda &\in \mathbb{R}_+^T. \end{aligned}$$

As the number of columns is typically very large, we use a *column generation algorithm* to solve LM. Below $\{\pi_i\}_{i=1}^m$ denote the dual variables associated with the complicating constraints, and π_0 the dual variable for the constraint $\sum_{t=1}^T \lambda_t = 1$, known as the *convexity constraint*.

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 an optimization problem over the set X .

Initialization. We suppose that a subset of points z^1, \dots, z^p is available leading to the Restricted LP Master Problem:

$$(RLM) \quad \begin{aligned} Z_{\text{RLM}} &= \max \sum_{t=1}^p (cz^t)\lambda_t \\ \sum_{t=1}^p (Az^t)\lambda_t &\leq b \\ \sum_{t=1}^p \lambda_t &= 1 \\ \lambda &\in \mathbb{R}_+^p. \end{aligned}$$

Solving RLM gives an optimal primal solution λ^* and an optimal dual solution $(\pi^*, \pi_0^*) \in \mathbb{R}_+^m \times \mathbb{R}^1$.

Primal Feasibility. Any feasible solution of RLM is feasible for LM. In particular, λ^* is a feasible solution of LM and so $Z_{\text{RLM}} = \sum_{t=1}^p (cz^t)\lambda_t^* = \pi^*b + \pi_0^* \leq Z_{\text{LM}}$.

Optimality Check for LM. We need to check whether (π^*, π_0^*) is dual feasible for LM.

This involves checking for each column, that is for each $x \in \text{conv}(X)$, whether the reduced price $cx - \pi^*Ax - \pi_0^* \leq 0$. Rather than examining each point separately, we treat all points in X implicitly by solving the optimization subproblem:

$$(SP) \quad \zeta^p = \max\{(c - \pi^*A)x - \pi_0^* : x \in X\}$$

with optimal solution x^* .

Stopping Criterion. If $\zeta^p = 0$, the solution (π^*, π_0^*) is dual feasible for LM, and so $Z_{LM} \leq \sum_{i=1}^m \pi_i^* b_i + \pi_0^*$. As the value of the primal feasible solution λ^* equals that of this upper bound, λ^* is optimal for LM.

Generating a New Column. If $\zeta^p > 0$, the column corresponding to the optimal solution x^* of the subproblem has positive reduced price. Setting $z^{p+1} = x^*$ and adding the column $(cz^{p+1}, Az^{p+1}, 1)^T$ leads to a new Restricted Linear Programming Master Problem that can be easily re-optimized (e.g. by the primal simplex algorithm).

A Dual (Upper) Bound. From the solution of the subproblem, we have that $\zeta^p \geq (c - \pi^* A)x - \pi_0^*$ for all $x \in X$. It follows that $(c - \pi^* A)x - \pi_0^* - \zeta^p \leq 0$ for all $x \in X$. Therefore, $(\pi^*, \pi_0^* + \zeta^p)$ is dual feasible in LM. Therefore,

$$Z_{LM} \leq \pi^* b + \pi_0^* + \zeta^p.$$

These different observations lead directly to an algorithm for LM that terminates when $\zeta^p = 0$. However, as in Lagrangian relaxation, it may be possible to terminate earlier.

An Alternative Stopping Criterion. If $\zeta^p > 0$, but the subproblem solution x^* satisfies the complicating constraints and $\pi^*(Ax^* - b) = 0$, then x^* is an optimal solution of LM. This follows because $\zeta^p = (c - \pi^* A)x^* - \pi_0^*$ implies that $cx^* = \pi^* Ax^* + \pi_0^* + \zeta^p = \pi^* b + \pi_0^* + \zeta^p$. Therefore, the primal feasible solution x^* has the same value as the upper bound on Z_{LM} .

Finding an Initial Feasible Solution. Introduce a phase 1 procedure in order to find an initial basic feasible solution of RLM. One approach is to introduce m artificial variables, or else generate m initial extreme points and then add one artificial variable with column $(b, 1)^T$ and a large cost M .

The Objective Value of the LP Master Problem. By construction $Z_{LM} = \max\{cx : Ax \leq b, x \in \text{conv}(X)\}$ giving the same upper bound as that obtained using Lagrangian relaxation of the complicating constraints $Ax \leq b$.

For the Resource-Constrained Shortest Path problem appearing in the Introduction, the polyhedron described by the flow constraints is integral. So treating these as subproblem constraints and the knapsack constraint as the complicating constraint, the Master LP bound will just be that of the linear programming relaxation of the original formulation. On the other hand, if the knapsack constraint appears in the subproblem, the upper bound will be strengthened with the convex hull of solutions of the knapsack constraint. The strength of the bounds for different decompositions of the GAP problem was discussed in Section 10.5.

Example 11.1 To demonstrate the column generation approach to solving the linear programming Master Problem, we consider a simple instance with $x \in \{0, 1\}^5$ and just two constraints. The first constraint is treated as the

complicating constraint and X is the set of 0–1 points satisfying the second constraint.

$$\begin{aligned} \max \quad & 6x_1 + 7x_2 + 4x_3 + 3x_4 + 2x_5 \\ & 5x_1 + 8x_2 + 6x_3 + 4x_4 + 2x_5 \leq 8 \\ & 7x_1 + 8x_2 + 6x_3 + 3x_4 + 3x_5 \leq 10 \\ x \quad & \in \{0, 1\}^5. \end{aligned}$$

We suppose that initially three points $z^t = e_t$ for $t = 1, 2, 3$ are available. Therefore, the restricted linear programming Master Problem is as follows:

$$\begin{aligned} \max \quad & 6\lambda_1 + 7\lambda_2 + 4\lambda_3 \\ & 5\lambda_1 + 8\lambda_2 + 6\lambda_3 \leq 8 \\ & \lambda_1 + \lambda_2 + \lambda_3 = 1 \\ \lambda \quad & \in \mathbb{R}_+^3. \end{aligned}$$

Iteration 1.

RLM. The primal solution is $Z^* = 7$, $\lambda^* = (0, 1, 0)$ and the dual solution $(\pi^*, \pi_0^*) = (0, 7)$.

The subproblem SP is

$$\begin{aligned} \zeta^1 = \max \quad & 6x_1 + 7x_2 + 4x_3 + 3x_4 + 2x_5 - 7 \\ & 7x_1 + 8x_2 + 6x_3 + 3x_4 + 3x_5 \leq 10 \\ x \quad & \in \{0, 1\}^5. \end{aligned}$$

with optimal solution $\zeta^1 = 2$, $x^* = z^4 = (1, 0, 0, 1, 0)$. As $\zeta^1 > 0$, $cz^4 = 9$, and $Az^4 = 9$, the column $(9, 9, 1)^T$ is added to the RLM.

Iteration 2.

RLM. The primal solution is $Z^* = 8.25$, $\lambda^* = \left(0, 0, \frac{1}{4}, \frac{3}{4}\right)$ and the dual solution $(\pi^*, \pi_0^*) = \left(\frac{3}{4}, \frac{9}{4}\right)$.

SP. The objective function of the subproblem is now $(c - \pi^* A)x - \pi_0 = (6, 7, 4, 3, 2)x - 0.75(5, 8, 6, 4, 2)x - 2.25 = 2.25x_1 + 1x_2 - 0.5x_3 + 0x_4 + 0.5x_5 - 2.25$.

Optimal solution $\zeta^2 = \frac{1}{2}$, $x^* = z^5 = (1, 0, 0, 0, 1)$. As $\zeta^2 > 0$, the column $(8, 7, 1)^T$ is added to the RLM.

Iteration 3.

RLM. The primal solution is $Z^* = 8.5$, $\lambda^* = \left(0, 0, 0, \frac{1}{2}, \frac{1}{2}\right)$ and the dual solution $(\pi^*, \pi_0^*) = \left(\frac{1}{2}, \frac{9}{2}\right)$.

SP. Optimal solution $\zeta^3 = 0$.

As $\zeta^3 = 0$, LM is solved with optimal solution $x = \frac{1}{2}z^4 + \frac{1}{2}z^5 = \frac{1}{2}(1, 0, 0, 1, 0) + \frac{1}{2}(1, 0, 0, 0, 1) = \left(1, 0, 0, \frac{1}{2}, \frac{1}{2}\right)$.

As the solution is not integer, the original problem is not yet solved and we have an upper bound of 8.5 on the optimal value. \square

11.4 Solving the Master Problem: Branch-and-Price

When the solution of the linear programming relaxation of the Master Problem LM is not integer, it is necessary to resort to branch and bound, possibly combined with cutting planes. Thus, at each node of the branch and bound tree, we need to decide

- (i) how to branch and
- (ii) how to solve the linear programming Master Problem LM.

We assume that the linear programming relaxation of the Master problem LM (11.3)–(11.7) has been solved with optimal solution $(\tilde{x}, \tilde{\lambda})$.

Branching at a Node

In the case in which $X \subseteq \{0, 1\}^n$, $\tilde{x} \notin \{0, 1\}^n$ if and only if $\tilde{\lambda} \notin \{0, 1\}^T$. This suggests branching on either the x or the λ variables.

Selecting the x variables leads to the branching scheme shown in Figure 11.1a, in which the set S of all feasible solutions is split into $S_0 = S \cap \{x : x_j = 0\}$ and $S_1 = S \cap \{x : x_j = 1\}$. Note that this is exactly the same type of scheme used in the basic branch-and-bound algorithm in Chapter 7.

The alternative is to branch on some fractional λ_t variable, fixing it to 0 and 1 respectively, see Figure 11.1b. Note, however, that on the branch in which $\lambda_t = 0$, just one column, corresponding to the solution z^t of the subproblem, 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, the new subproblems are not easy to solve.

In the case in which $X \subseteq \mathbb{Z}^n$ and $\{z^t\}_{t=1}^T$ are the extreme points of $\text{conv}(X)$, setting $\lambda_t \in \{0, 1\}$ excludes some points of X , so again branching on x variables is the better option.

So from now on, we will branch on x variables whenever possible. Later, we consider models with identical subproblems in which the x variables are aggregate

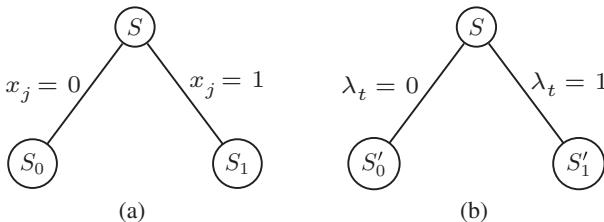


Figure 11.1 Branching for 0-1 column generation: (a) original and (b) column variables.

variables. In such cases, branching on the x -variables may not be sufficient to ensure that we can recover the solution of the original problem.

Solving the LP Master at a Node

A node in the branch-and-bound tree is defined by a series of bound constraints $\ell \leq x \leq u$ and possibly more general branching constraints or cutting planes. Let $Cx \leq g$ denote all the additional constraints associated to the node so that the associated problem at the node is $\max\{cx : Ax \leq b, Cx \leq g, x \in X\}$. There is now a choice: Which of the new constraints should we treat as complicating constraints and which should we use to define the new set X arising in the subproblem?

Suppose that constraints $C^1x \leq g^1$ are treated as complicating constraints with associated dual variables μ and $C^2x \leq g^2$ are added to the set X where $C = (C^1, C^2)^T$ and $g = (g^1, g^2)^T$. The columns of RLM now take the form $(cz^t, Az^t, C^1z^t, 1)^T$ and otherwise the treatment of the Restricted Master Problem is unchanged.

However, if (C^2, g^2) is nonempty, the subproblem now takes the form:

$$\max\{(c - \pi^*A - \mu^*C^1)x - \pi_0^* : x \in X \cap \{x : C^2x \leq g^2\}\}.$$

Two factors come into play in deciding how to treat the new constraints.

Complexity of the Subproblem. Optimizing over the set X is supposed to be relatively easy. Optimizing over $X \cap \{x : C^2x \leq g^2\}$ may still be easy in which case we talk of *robust* column generation, or it may be much more difficult.

As an example, if the set X is a 0–1 knapsack set, adding bound constraints $x_j = 0$ or $x_j = 1$ to X will again lead to a 0–1 knapsack set, so the same algorithm for solving 0–1 knapsack problems can be used at all the nodes of the tree. On the other hand, adding a general constraint $fx \leq g$ leads to a new set with two constraints for which the 0–1 knapsack algorithm used at the top node will no longer work.

Strength of the Linear Programming Bound. We have seen above that the value of the solution of the LP Master is $\max\{cx : Ax \leq b, C^1x \leq g^1, x \in \text{conv}(X \cap \{x : C^2x \leq g^2\})\}$. So the bound becomes potentially stronger as constraints are transferred from $C^1x \leq g^1$ to $C^2x \leq g^2$. In particular, $\max\{cx : Ax \leq b, Cx \leq g, x \in \text{conv}(X)\} \geq \max\{cx : Ax \leq b, x \in \text{conv}(X \cap \{x : Cx \leq g\})\} \geq Z_M$.

Note that, if one just adds a single bound constraint, adding it to the set X may lead to a better upper bound than when adding it to the complicating constraints $Ax \leq b$, but the subproblem may become more difficult to solve.

Example 11.2 Continuing with Example 11.1, the optimal solution of the linear programming Master Problem is $x = \left(1, 0, 0, \frac{1}{2}, \frac{1}{2}\right) \notin \{0, 1\}^5$.

Branch-and-Price. Here we choose to branch on the original x variables. As the column generation subproblem is a 0–1 knapsack problem, we can add the branching constraints in the column generation subproblem.

Node 1. $Z_{LM} = 8.5$. $x = \left(1, 0, 0, \frac{1}{2}, \frac{1}{2}\right)$. We branch on one of the integer variables taking a fractional value, for example x_4 .

Node 2. Branch $x_4 = 0$.

As $x_4^4 > 0$, one can set $\lambda_4 = 0$.

Iteration 1.

RLM. $Z_{RLM} = 8$, $\lambda^* = (0, 0, 0, 0, 1)$, $(\pi^*, \pi_0^*) = (0, 8)$.

SP with $x_4 = 0$. $\zeta = 0$. LM is solved.

As $x = z^5 \lambda_5^*$ is a feasible integer solution. New incumbent $z^5 = (1, 0, 0, 0, 1)$ of value 8. Node pruned by optimality.

Node 3. Branch $x_4 = 1$. As $\sum_{t=1}^5 \lambda_t = 1$ and $x_4 = \sum_{t=1}^5 z_4^t \lambda_t = \lambda_4 = 1$, set $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_5 = 0$.

Iteration 1.

RLM. Problem infeasible.

Phase 1. Introduce a new artificial column with cost $M = 100$. The resulting RLM is

$$\begin{aligned} \max \left\{ \sum_{t=1}^p (cz^t) \lambda_t - My, \sum_{t=1}^p (Az^t) \lambda_t + by \leq b, \right. \\ \left. \sum_{t=1}^p \lambda_t + y = 1, \lambda \in \mathbb{R}_+^p, y \in \mathbb{R}_+^1 \right\}. \end{aligned}$$

RLM $Z_{RLM} = -100$, $\lambda = 0$, $y = 1$, $(\pi, \pi_0) = (109, -972)$.

SP with $x_4 = 1$. $\zeta = 539$, $x = z^5 = (0, 0, 0, 1, 0)$.

Iteration 2.

RLM $Z_{RLM} = 7.8$, $\lambda = (0, 0, 0, 0.8, 0, 0.2)$, $(\pi, \pi_0) = (1.2, -1.8)$

SP with $x_4 = 1$. $\zeta = 0.4$, $z^6 = (0, 0, 1, 1, 0)$

Iteration 3.

RLM $Z_{RLM} = 8.0$, $\lambda = (0, 0, 0, 0.5, 0, 0, 0.5)$, $(\pi, \pi_0) = (1, 0)$

SP with $x_4 = 1$. $\zeta = 0$.

$Z_{LM} = 8$. Node pruned by bound.

All nodes are pruned. $x^* = (1, 0, 0, 0, 1)$ is optimal with $Z_M = Z_{LM} = 8$. □

Pursuing this example, we now consider what happens when cuts are added in the course of the algorithm.

Example 11.3 Taking the fractional solution of LM at node 1, we observe that the 0–1 knapsack separation heuristic of Chapter 9 applied to the “complicating constraint” $5x_1 + 8x_2 + 6x_3 + 4x_4 + 2x_5 \leq 8$ generates the violated extended cover

inequality $x_1 + x_2 + x_4 \leq 1$. Here we consider two cases: in the first case (A), this inequality is added as a “complicating constraint,” and in the second case (B), it is added as a “subproblem constraint.”

Branch-Cut-and-Price (A: Cut Added to Complicating Constraints)

Node 1. $Z_{\text{RLM}} = 8.5$. $x = \left(1, 0, 0, \frac{1}{2}, \frac{1}{2}\right)$.

Adding $x_1 + x_2 + x_4 \leq 1$ to the complicating constraints, the RLM has an additional constraint $\sum_{t=1}^T (z_1^t + z_2^t + z_4^t) \lambda_t = \lambda_1 + \lambda_2 + 2\lambda_4 + \lambda_5 \leq 1$ with dual variable $\hat{\pi}$.

RLM $Z_{\text{LM}} = 8$, $\lambda = (0, 0, 0, 0, 1)$. $x = (1, 0, 0, 0, 1)$. $\pi = 0$, $\hat{\pi} = 1$, $\pi_0 = 7$.

The form of the subproblem is unchanged.

SP $\zeta = 0$. LM is solved.

LM has an optimal LP solution that is integer and is thus optimal.

Branch-Cut-and-Price (B: Cut Added to Subproblem)

Node 1. We take RLM from Example 11.1 with $Z_{\text{RLM}} = 8.5$. $x = \left(1, 0, 0, \frac{1}{2}, \frac{1}{2}\right)$.

$(\pi^*, \pi_0^*) = (0.5, 4.5)$. As x^4 is no longer feasible in $X \cap \{x : x_1 + x_2 + x_4 \leq 1\}$, we can remove x^4 by setting $\lambda_4 = 0$. RLM which is otherwise unchanged is resolved.

RLM $Z_{\text{LM}} = 8$, $\lambda = (0, 0, 0, 0, 1)$. $\pi^* = 0$, $\pi_0^* = 8$. $x = (1, 0, 0, 0, 1)$ is feasible for M with value 8.

SP is now of the form: $\zeta = \max\{(c - \pi^* A)x - \pi_0^* : x_1 + x_2 + x_4 \leq 1, x \in X\}$.

SP $\zeta = 0$. LM is solved.

LM has an optimal LP solution that is integer and is thus optimal. \square

11.5 Problem Variants

11.5.1 Handling Multiple Subproblems

For problems in the form IP with $K > 1$ with $\{z^{kt}\}_{t=1}^{p^k}$ the extreme points of X^k for $k = 1, \dots, K$, similar transformations lead to a Master Problem of the form:

$$\begin{aligned} Z_M = \max & \sum_{k=1}^K \sum_{t=1}^{p^k} (c^k z^{kt}) \lambda_t^k \\ & \sum_{k=1}^K \sum_{t=1}^{p^k} (A^k z^{kt}) \lambda_t^k \leq b \\ & \sum_{t=1}^{p^k} \lambda_t^k = 1 \quad \text{for } k = 1, \dots, K \\ & \lambda^k \in \mathbb{R}_+^{p^k} \quad \text{for } k = 1, \dots, K \\ & x^k - \sum_{t=1}^{p^k} z^{kt} \lambda_t^k = 0 \quad \text{for } k = 1, \dots, K \\ & x^k \in \mathbb{Z}^{n^k} \quad \text{for } k = 1, \dots, K. \end{aligned}$$

The lower bound on Z_{LM} is $Z_{\text{RLM}} = \sum_{k=1}^K \sum_{t=1}^{p^k} (c^k z^{kt}) \lambda_t^k$.

In the RLM, the k -th convexity constraint has dual variable π_{k0} .

The subproblems take the form: $\zeta^k = \max\{(c^k - \pi A^k)x^k - \pi_{k0} : x^k \in X^k\}$.

The linear program is solved when $\zeta^k = 0$ for $k = 1, \dots, K$.

An upper bound on Z_{LM} is $Z_{\text{RLM}} + \sum_{k=1}^K \zeta^k$.

An extreme alternative is to treat the set $X = X^1 \times \dots \times X^K$ as a single subproblem set. In this case, the columns of the Master take the form $(\sum_k c^k z^{k,t_k}, \sum_k A^k z^{k,t_k}, 1)$, and there is a single convexity constraint.

11.5.2 Partitioning/Packing Problems with Additional Variables

Many problems involve complicating constraints that are Partitioning/Packing constraints on a subset of the variables. Such problems take the form

$$Z = \max \left\{ \sum_{k=1}^K (c^k x^k + f^k w^k), \sum_{k=1}^K x^k \leq \mathbf{1}, (x^k, w^k) \in X^k, \text{ for } k = 1, \dots, K \right\}.$$

These problems can be treated as in the subsection 11.5.1. An example of such a problem is:

Multi-item Lot-Sizing. We are given demands d_u^k for items $k = 1, \dots, K$ over a time horizon $u = 1, \dots, n$. All items must be produced on a single machine. The machine can produce only one item in each period and has a capacity C^k per period if item k is produced in the period. Given production, storage, and setup costs (p_u^k, h_u^k, c_u^k) for each item in each period, we wish to find a minimum cost production plan. Letting $x_u^k = 1$ if item k is set-up in period u and 0 otherwise, y_u^k be the amount of item k produced in period u and s_u^k be the amount of item k in stock at the end of period u , the problem can be formulated as:

$$\begin{aligned} \min & \sum_{k=1}^K \sum_{u=1}^n (p_u^k y_u^k + h_u^k s_u^k + c_u^k x_u^k) \\ & \sum_{k=1}^K x_u^k \leq 1 \quad \text{for } u = 1, \dots, n \\ & (x^k, y^k, s^k) \in X^k \quad \text{for } k = 1, \dots, K, \end{aligned}$$

with X^k given by:

$$\begin{aligned} s_{u-1}^k + y_u^k &= d_u^k + s_u^k \quad \text{for } u = 1, \dots, n \\ y_u^k &\leq C_u^k x_u^k \quad \text{for } u = 1, \dots, n \\ s_0^k &= 0 \\ s^k &\in \mathbb{R}_+^{n+1}, y^k \in \mathbb{R}_+^n, x^k \in \{0, 1\}^n. \end{aligned}$$

11.5.3 Partitioning/Packing Problems with Identical Subsets

Here we suppose that the problem is of the form $X^k = X \subset \{0, 1\}^m$, $c^k = c$, $A^k = I_m$ for all k and $b = \mathbf{1}$. With $z^{kt} = z^t$ for all k, t and $\mu_t = \sum_k \lambda_t^k$, the Master problem can be rewritten in the form:

$$(MIS) \quad \begin{aligned} Z_M &= \max \sum_{t=1}^T (cz^t) \mu_t \\ &\sum_{t: z_i^t = 1} \mu_t \leq 1 \quad \text{for } i = 1, \dots, m \\ &\sum_{t=1}^T \mu_t = K \\ &\mu \in \{0, 1\}^T. \end{aligned}$$

There is now just a single column generation problem:

$$\zeta = \max \{(c - \pi)x - \pi_0 : x \in X\}.$$

An example of such a problem is:

A Clustering Problem. Given a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, edge costs f_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). The problem can be formulated as

$$\begin{aligned} &\max \sum_{k=1}^K \sum_{e \in E} f_e w_e^k \\ &\sum_{k=1}^K x_i^k = 1 \quad \text{for } i \in V \\ &(w^k, x^k) \in X \quad \text{for } k = 1, \dots, K \end{aligned}$$

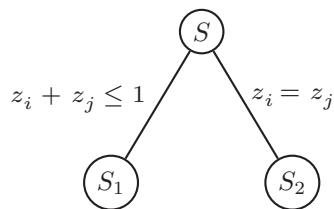
where $X = \{(w, x) \in \{0, 1\}^m \times \{0, 1\}^n : w_e \leq x_i, w_e \leq x_j, w_e \geq x_i + x_j - 1 \text{ for } e = (i, j) \in E, \sum_{i \in V} d_i x_i \leq C\}$ with $x_i = 1$ if node i is in the cluster and $w_e = 1$ if edge e has both endpoints in the cluster.

Solving the linear programming relaxation of MIS is essentially as before. The first question that arises is:

How to branch when the solution μ^* of its linear programming relaxation is not integral?

It is now not at all obvious how to recover the original variables x^k or the λ_t^k variables, so the branching scheme proposed in Section 11.4 must be modified. We now consider two possibilities.

Figure 11.2 A branching scheme for a partitioning problem.



Branching Rules

- (i) If $\sum_{t=1}^T \mu_t = \alpha \notin Z$, then form two branches with $\sum_{t=1}^T \mu_t \leq \lfloor \alpha \rfloor$ and $\sum_{t=1}^T \mu_t \geq \lceil \alpha \rceil$, respectively.
- (ii) A second possibility is based on the simple observation that if we consider a pair of rows each 0–1 column z^t either has a 1 appearing in both rows, or not. So we choose a pair of elements (rows) i and j with $1 \leq i < j \leq m$ for which

$$0 < \sum_{t: z_i^t = z_j^t = 1} \mu_t < 1,$$

and we then form two branches with $\sum_{t: z_i^t = z_j^t = 1} \mu_t = 1$ and $\sum_{t: z_i^t = z_j^t = 1} \mu_t = 0$, respectively.

On the first branch, we impose that i and j lie in the same subset, and on the second that i and j lie in different subsets. In the first case, all columns corresponding to subsets Q containing either i or j but not both are eliminated from the Master Problem and the constraint $z_i = z_j$ is added to the subproblem to ensure that any new column generated does not contain i but not j , or vice versa. In the second case, columns containing both i and j are eliminated from the Master, and the constraint $x_i + x_j \leq 1$ is added to the subproblem, see Figure 11.2.

The following result says that this second branching scheme is sufficient.

Proposition 11.1 *If $\mu^* \notin \{0, 1\}^T$, there exist rows i, j with $1 \leq i < j \leq m$ such that*

$$0 < \sum_{t: z_i^t = z_j^t = 1} \mu_t < 1.$$

Such a pair is also not difficult to find.

11.6 Computational Issues

To make a column generation or branch-and-price algorithm efficient, there are many ideas of ways to speed up the solution process. Here, we list a few of them.

Speeding Up the Solution of the LP Master.

It is well known that column generation algorithms can take many iterations initially before generating good dual variables/prices and thus interesting columns. Secondly that there are tailing-off effects with very many iterations needed to complete convergence.

- i) To tackle the start-up problems, it is usual to generate heuristically a good set of starting columns.
- ii) Various ways are used to stabilize the dual variables either by adding constraints on the dual variables, restricting them to a certain region and/or using penalty functions.
- iii) Heuristics can be used to find good, but not optimal, solutions to the subproblems. The subproblems only need to be solved exactly on the final iteration in order to prove that the LP Master Problem (LM) is solved.

Speeding Up the Solution of the IP Master.

- i) As upper bounds on the value of Z_{LM} are generated, these can be used for possible pruning, so it is not necessary solve LM to optimality at each node of the branch-and-bound tree.
- ii) Strong branching is recommended.
- iii) Development of special primal heuristics for column generation problems.
- iv) Possible use of complete enumeration when the duality gap at a node is very small.
- v) Choice of branching variables.

Branch-Cut-and-Price.

- i) Development of cutting planes or extended formulations in order to tighten the bounds.
- ii) Use of robust cuts if possible to avoid complicating the subproblems.
- iii) Find fast ways to use nonrobust cuts defined over the λ variables.

11.7 Branch-Cut-and-Price: An Example*

Here we present a branch-cut-and-price algorithm for a more complicated problem, the symmetric Capacitated Vehicle Routing Problem (CVRP).

11.7.1 A Capacitated Vehicle Routing Problem

Given a complete graph $G = (V, E)$ with node set $V = \{0, 1, \dots, n\}$ consisting of a depot at node 0 and nodes $1, \dots, n$ representing the clients, c_e is the cost of traversing edge $e \in E$. There are K identical vehicles of capacity Q and each client i has a demand d_i . The problem is to assign subtours starting and finishing at the depot to each vehicle, so that each client receives his demand d_i from a single vehicle,

the capacity of a vehicle is not exceeded and the total sum of the edges traversed by the vehicles is minimized.

A first formulation is close to the subtour formulation of the traveling salesman problem. Let $x_e = 1$ if the edge e is traversed by one of the vehicles. This leads to the formulation:

$$\min \sum_{e \in E} c_e x_e \quad (11.9)$$

$$\sum_{e \in \delta(0)} x_e = 2K \quad (11.10)$$

$$\sum_{e \in \delta(i)} x_e = 2 \quad \text{for } i = 1, \dots, n \quad (11.11)$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \lceil \frac{d(S)}{Q} \rceil \quad \text{for } S \subset V, 0 \notin S \quad (11.12)$$

$$x_e \in \{0, 1\} \quad \text{for } e \in E \setminus \delta\{0\} \quad (11.13)$$

$$x_e \in \{0, 1, 2\} \quad \text{for } e \in \delta\{0\} \quad (11.14)$$

where $\delta(i)$ denotes the edges incident to node i , $\delta(S)$ the edges in the cut separating S and $V \setminus S$ and $d(S) = \sum_{i \in S} d_i$. Here (11.10) forces K vehicles to leave and enter the depot, (11.11) forces a single vehicle entering and leaving each client and (11.12) ensures that the capacity of the vehicles arriving at a set S of clients is sufficient to satisfy their demands. Note that this formulation has an exponential number of constraints.

A second formulation views the problem as a partitioning problem in which columns ideally represent the subtours of a vehicle. However, as finding an optimal subtour even for a single vehicle is an \mathcal{NP} -hard problem, the idea is to use a relaxation of a tour over which it is possible to optimize. Specifically, a q -route is a walk (allowing more than one visit to a client) in which the load d_i is counted at each visit to client i and the vehicle capacity is not exceeded. We call a q -route without two-cycles (the sequence of successive visits i, j, i is excluded) a $q2$ -route. Let \mathcal{R} be the set of all $q2$ -routes and let variable $\lambda_t = 1$ if $q2$ -route t is used by one of the vehicles. This leads to the formulation:

$$\min \sum_{t \in \mathcal{R}} c_t \lambda_t \quad (11.15)$$

$$\sum_{t \in \mathcal{R}} a_{0,t} \lambda_t = 2K \quad (11.16)$$

$$\sum_{t \in \mathcal{R}} a_{i,t} \lambda_t = 2 \quad \text{for } i = 1, \dots, n \quad (11.17)$$

$$\lambda_t \in \{0, 1\} \quad \text{for } t \in \mathcal{R}, \quad (11.18)$$

where c_t is the travel cost of $q2$ -route t and $a_{i,t}$ is the number of edges incident to node i in the $q2$ -route t . The constraints (11.16) and (11.17) ensure that each client is visited exactly once on the K $q2$ -routes and thus that the resulting solution provides a set of feasible vehicle routes.

Neither of the two formulations P_1 and P_2 obtained by dropping the integrality constraints in these two integer programs contains the other, so a stronger formulation is obtained by intersecting the two. Letting $\rho_{e,t}$ be the number of occurrences of edge e in $q2$ -route t , it follows that $x_e = \sum_{t \in \mathcal{R}} \rho_{e,t} \lambda_t$. Substituting for x_e gives a third integer program:

$$\begin{aligned} \min & \sum_{t \in \mathcal{R}} c_t \lambda_t \\ \text{s.t.} & \sum_{t \in \mathcal{R}} a_{0,t} \lambda_t = 2K \\ & \sum_{t \in \mathcal{R}} a_{i,t} \lambda_t = 2 \quad \text{for } i = 1, \dots, n \\ & \sum_{t \in \mathcal{R}} A_{S,t} \lambda_t \geq \lceil \frac{d(S)}{Q} \rceil \quad \text{for } S \subset V, 0 \notin S \\ & \lambda_t \leq 1 \quad \text{for } t \in \mathcal{R} \\ & \lambda \in \mathbb{Z}^{\mathcal{R}}, \end{aligned}$$

where $A_{S,t}$ is the number of edges in $q2$ -route t in the cut-set $\delta(S)$.

The basic algorithm is now straightforward.

1. Consider the Relaxed Master problem involving a subset $\mathcal{R} = \{1, \dots, r\}$ of columns and subsets S_1, \dots, S_p .
2. After solving the linear program with dual variables $\alpha \in \mathbb{R}^1, \beta \in \mathbb{R}^n, \gamma \in \mathbb{R}_+^p, \mu \in \mathbb{R}_-^r$, solve the subproblem

$$\zeta = \min \left\{ \sum_{e \in E} \bar{c}_e x_e : x \text{ the incidence vector of a } q2\text{-route} \right\} \quad (11.19)$$

where the reduced cost \bar{c}_e of x_e is given by

$$\bar{c}_e = c_e - \beta_i - \beta_j - \sum_{p \in [1,q]: e \in \delta(S_p)} \mu_p \quad \text{if } e = (i,j) \notin \delta(0) \quad \text{and} \quad \bar{c}_e = c_e - \alpha - \beta_j - \sum_{p \in [1,q]: e \in \delta(S_p)} \mu_p \quad \text{if } e = (0,j) \in \delta(0).$$

As long as $\zeta > 0$, new columns are added to the problem. When $\zeta = 0$, the linear programming relaxation with all the $q2$ -routes (11.9)–(11.14) has been solved.

3. Now we test whether the corresponding solution $x_e^* = \sum_{t \in \mathcal{R}} \rho(e, t) \lambda_t^*$ satisfies the capacitated subtour constraints. For this, the separation problem is:

$$\phi = \min_{S \subset V, 0 \notin S} \left\{ \sum_{e \in \delta(S)} x_e^* - \lceil \frac{d(S)}{Q} \rceil \right\}.$$

If $\phi < 0$, a new capacitated subtour constraint is added and we return to 1. with at least all the columns ($q2$ -routes) in the basis and all the subtour constraints that are tight.

If $\phi = 0$, the relaxed Master problem is solved. If the corresponding $x^* \in \mathbb{Z}_+^{|V|}$, the problem is solved, otherwise one needs to branch.

4. Rather than branching on simple bound constraints, it is suggested to branch on constraints of the form $\sum_{e \in \delta(S)} x_e \leq 2s$ and $\sum_{e \in \delta(S)} x_e \geq 2s + 2$. As these have the same form as the capacitated subtour constraints, the resulting column generation subproblem is essentially unchanged.

11.7.2 Solving the Subproblems

Column Generation

Here we consider the column generation problem (11.19) of finding a $q2$ -route to enter the basis. A simple dynamic programming algorithm is

$$H(j, k, q) = \min_{i \in V \setminus \{j, k\}} \{H(i, j, q - d_j) + \bar{c}_{j,k}\} \quad j, k \in V, j \neq k, d_j \leq q \leq Q$$

where $H(j, k, q)$ is the minimum cost of a walk without two-cycles starting at node 0 and ending with node j followed by node k with the vehicle carrying load q on edge (j, k) . The minimum cost $q2$ -route then has cost

$$\min_{j, k \in V: j \neq k} \min_{q: d_k \leq q \leq Q} \{H(j, k, q - d_k) + \bar{c}_{0k}\}.$$

A fairly recent improvement is to replace q -routes by ng -routes. For each node $i \in V \setminus \{0\}$, $N(i) \subset V$ with $i \in N(i)$ is a set of nodes, called neighbors. An ng -route is a walk beginning and ending at node 0 in which node i can only be revisited if at least one node j visited after node i is such that $j \notin N(i)$. Let $G(i, q, S)$ denote a minimum cost ng -walk starting at node 0 and arriving at node i having delivered exactly an amount q with S the set of nodes that are forbidden as immediate successors of i . Consider a walk arriving at node i with excluded successors S and suppose that j is the next node on the path. If $k \in S$ with $k \neq j$, k cannot be a successor of j unless $k \notin N(j)$. This leads to the dynamic programming recursion

$$G(j, q, (S \cap N(j)) \cup \{j\}) = \min_{i \in V} \{G(i, q - d_j, S) + \bar{c}_{ij}\}.$$

Cut Generation

Checking whether one of the capacity inequalities (11.12) is violated is an \mathcal{NP} -hard problem. Here we present the idea behind a couple of heuristics and also show how the separation problem can be formulated exactly as a mixed integer program. The idea of the first heuristic is very simple. Given the fractional solution x^* , consider the support graph $G^* = (V^*, E^*)$ where $V^* = V \setminus \{0\}$ and $e = (i, j) \in E^*$ if $x_e^* > 0$. Let S_1, \dots, S_r be the components of G^* that are not

connected by any flow to the depot. Test the capacity(GSEC) inequality (11.12) for S_i for $i = 1, \dots, r$. Another heuristic is based on the fact that if the inequality is weakened to have an rhs of $\frac{d(S)}{Q}$ instead of $\lceil \frac{d(S)}{Q} \rceil$, the separation problem is polynomial as for GSECs. In practice other heuristics are also used to find violated capacity inequalities and to try to find violations of other classes of inequalities that are known to be valid for CVRP.

The MIP for the separation of the capacity inequalities (11.12) is only used as a last resort or to test the effectiveness of the heuristics. Using the variables $z_i = 1$ if $i \in S$ and 0 otherwise, $w_e = 1$ if edge $e = (i, j) \in \delta(S)$ and 0 otherwise and $\phi \in \mathbb{Z}_+^1$, the resulting MIP formulation is:

$$\begin{aligned} \min \sum_{e \in E} x_{ij}^* w_{ij} - 2(\phi + 1) \\ w_{ij} &\geq z_j - z_i \quad \text{for } (i, j) \in E \\ w_{ij} &\geq z_i - z_j \quad \text{for } (i, j) \in E \\ \left(\sum_{i \in V \setminus \{0\}} d_i z_i \right) / Q &\geq \phi + 1 \\ z_0 = 0, z &\in \{0, 1\}^{|V|}, w \in \mathbb{R}_+^{|E|}, \phi \in \mathbb{Z}_+^1 \end{aligned}$$

with ϵ very small and positive. The first pair of constraints ensure that the w variables define a cut-set $\delta(S)$, and the third that $\phi + 1 \leq \lceil \frac{\sum_{i \in S} d_i}{Q} \rceil$.

11.7.3 The Load Formulation

Here we introduce an alternative formulation in which the travel times on an arc depend on the direction traveled. The demands and vehicle capacity are assumed to be integer. $D_Q = (V, A_Q)$ is a multigraph with nodes $V = \{0, 1, \dots, n\}$ as above and arcs $a_q = (i, j, q)$ for $d_i \leq q \leq Q$ joining i to j . Let $x_{ij}^q = 1$ if some vehicle goes from node i to node j carrying a load of q units and 0 otherwise.

The formulation is

$$\min \sum_{q=0}^Q \sum_{(i,j) \in A} c_{ij} x_{ij}^q \tag{11.20}$$

$$\sum_{a \in \delta^+(0)} x_a^0 = K \tag{11.21}$$

$$\sum_{q=0}^Q \sum_{a \in \delta^+(i)} x_a^q = 1 \quad \text{for } i \in V \setminus \{0\} \tag{11.22}$$

$$\sum_{a \in \delta^-(i)} x_a^{q-d_i} - \sum_{a \in \delta^+(i)} x_a^q = 0 \quad \text{for } i = 1, \dots, n, d_i \leq q \leq Q \tag{11.23}$$

$$x_{ij}^q \in \{0, 1\} \quad \text{for } (i, j, q) \in A_Q. \tag{11.24}$$

Constraint (11.21) stipulates that K empty vehicles leave the depot, (11.22) that one vehicle visits each client, and (11.23) that the load of a vehicle increases by d_i when it visits client i . Note that the number of variables and nodes is pseudo-polynomial as it depends on the size of Q . Here again the next step is to introduce columns for q -routes or ng -routes. It is also possible to generate new families of cuts involving the arc-load variables. The steps to develop an effective BCP algorithm then include improved labeling algorithms for the column generation, heuristic pricing, reduced cost fixing, route enumeration, strong branching, robust cuts involving the new load variables, heuristic separation of other CVRP cuts, a special subclass of nonrobust cuts, etc.

11.8 Notes

Several books and surveys on column generation include the book of Desaulniers et al. [96] and a survey of Vanderbeck and Wolsey [282]

- 11.1 The fundamental paper on the decomposition of linear programs, known as Dantzig–Wolfe decomposition, appeared in [94].
- 11.3 The first use of column generation to solve the Master linear program arising from an IP problem is probably the work on the cutting stock problem of Gilmore and Gomory [141, 142]. The equivalence of the bounds provided by the linear programming Master and the Lagrangian dual has been known since Geoffrion [137].
- 11.4 The first papers on IP column generation in the 1980s were by Desrosiers et al. [99] and Desrosiers and Soumis [98] on routing problems in which the subproblems are constrained shortest path problems that are solved by dynamic programming. Several applications of branch-and-price were published in the 1990s such as work on a multi-item lot-sizing by Vanderbeck [279], on a clustering problem by Johnson et al. [188], on the GAP in Savelsbergh [263], and on binary and integer cutting stock problems in Vance et al. [278] and Vanderbeck [280] respectively.
- 11.5 The branching rule (ii) is from Ryan and Foster [261]. More general branching rules for problems with identical subproblems that are not restricted to 0–1 variables appear in Barnhart et al. [34] and Vanderbeck and Wolsey [281].
- 11.6 Stabilization techniques for the solution of the LP Master problem are treated in Pessoa et al. [247]. A primal heuristic is presented in Sadykov et al. [262]. The thesis of Gamrath [131] contains a detailed survey of the major aspects of branch-cut-and-price.
- 11.7 The presentation here is largely based on Fukasawa et al. [127]. The simple pricing DP for q -routes without two-cycles runs in $O(n^3Q)$, but a $O(n^2Q)$

version is given in Christofides et al. [69]. Many of the valid inequalities for CVRP along with heuristics for their separation are described in Letchford et al. [207] and Lysgaard et al. [216]. A software package with these separation routines for CVRP is available from Lysgaard [215].

The load formulation of Section 11.7.3 was introduced in Pessoa et al. [246]. In Pecin et al. [245], the load formulation is used. It appears to be the most effective formulation to date for the basic CVRP. The article includes many new ideas of the authors such as new robust cuts on the load variables, and of others such as *ng*-routes due to Baldacci et al. [29], total enumeration when the duality gap is sufficiently small in Baldacci et al. [28] and non-robust cuts in Jepsen et al. [183]. It is also shown in Pessoa et al. [247] that several variants such as problems with time-windows, heterogeneous fleets, etc. can be treated using a similar approach.

Information on a solver for CVRP and several related problems can be found in [248]. An alternative branch-cut-and-price algorithm is proposed in Costa et al. [83].

11.9 Exercises

- Consider the problem UFL with formulation as in Chapter 1

$$\begin{aligned} \min & \sum_{i=1}^m \sum_{j=1}^n c_{ij} y_{ij} + \sum_{j=1}^n f_j x_j \\ & \sum_{j=1}^n y_{ij} = 1 \quad \text{for } j = 1, \dots, n \\ & y_{ij} - x_j \leq 0 \quad \text{for } i = 1, \dots, m, j = 1, \dots, n \\ & y \in \mathbb{R}_+^{mn}, x \in \{0, 1\}^n. \end{aligned}$$

Consider column generation, taking the constraints $\sum_{j=1}^n y_{ij} = 1$ as the complicating constraints.

- Describe the extreme points of $\text{conv}(X)$ where $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^m : y_i \leq x \text{ for } i = 1, \dots, m\}$.
- What is the complete Master Problem?
- What is the column generation subproblem?
- Consider an instance with $m = 4, n = 3$,

$$(c_{ij}) = \begin{pmatrix} 2 & 1 & 5 \\ 3 & 4 & 2 \\ 6 & 4 & 1 \\ 1 & 3 & 7 \end{pmatrix} \text{ and } f = (8, 6, 5).$$

Construct the restricted Master Problem using all the initial columns in which a depot serves exactly two clients. Carry out an iteration of the algorithm.

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

$$\max \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \quad (11.25)$$

$$\sum_{j=1}^n x_{ij} = 1 \quad \text{for } i = 1, \dots, m \quad (11.26)$$

$$\sum_{i=1}^m a_{ij} x_{ij} \leq b_j \quad \text{for } j = 1, \dots, n \quad (11.27)$$

$$x \in \{0, 1\}^{mn}. \quad (11.28)$$

Solve an instance with $m = 3$, $n = 2$, $(c_{ij}) = \begin{pmatrix} 20 & 16 \\ 15 & 19 \\ 19 & 14 \end{pmatrix}$, $(a_{ij}) = \begin{pmatrix} 5 & 7 \\ 3 & 8 \\ 2 & 10 \end{pmatrix}$, and

$b = \begin{pmatrix} 6 \\ 21 \end{pmatrix}$ by column generation.

4. *Makespan on Parallel Machines.* Here one has K machines and m jobs. Each job must be carried out on one of the machines. The processing time of job i on machine k is a_{ik} . The objective is to complete all the jobs as soon as possible. The variables are $x_{ik} = 1$ if job i is assigned to machine k and 0 otherwise. η is the completion time of the last job to finish. Formulate as an MIP and discuss how this problem can be tackled by Branch-and-Price.
5. *The Cutting Stock Problem* (See Section 5.5). 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.
6. *The Bin Packing Problem.* K identical bins of size d are given as well m distinct items. Item i takes up a_i space in a bin. The problem is to fit all the items into as few bins as possible. Formulate and suggest a solution algorithm.

12

Benders' Algorithm

12.1 Introduction

Here we consider one approach for solving problems of the form

$$Z = \max\{cx + hy : Fx + Gy \leq d, x \in X \subseteq \mathbb{Z}_+^n, y \in \mathbb{R}_+^p\} \quad (12.1)$$

where in many cases p is large, while n is relatively small.

One example is the capacitated facility location problem in which each facility has a capacity d_j , each client has a demand a_i , there is a fixed cost f_j of using facility j , and a per unit transportation cost c_{ij} between facility j and client i . The goal is to minimize the total cost while satisfying the demands subject to the capacity constraints. This can be formulated as:

$$\min \sum_{j=1}^n f_j x_j + \sum_{i=1}^m \sum_{j=1}^n c_{ij} y_{ij}$$

$$\sum_{i=1}^m y_{ij} \leq d_j x_j \quad \text{for } j = 1, \dots, n$$

$$\sum_{j=1}^n y_{ij} = a_i \quad \text{for } i = 1, \dots, m$$

$$y_{ij} \leq \min(a_i, d_j) x_j \quad \text{for } i = 1, \dots, m, j = 1, \dots, n$$

$$\sum_{j=1}^n d_j x_j \geq \sum_{i=1}^m a_i$$

$$x \in \{0, 1\}^n, \quad y \in \mathbb{R}_+^{mn}$$

where $x_j = 1$ if facility j is used and y_{ij} is the amount shipped from facility j to client i .

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.

Companion website: www.wiley.com/go/wolsey/integerprogramming2e

Another example arises when there is an extended formulation $Q = \{(x, y) : Fx + Gy \leq d, x \in \mathbb{R}^n, y \in \mathbb{R}_+^p\}$ for some set $P \subset \mathbb{R}^n$, i.e. $P = \text{proj}_x(Q)$. Here the problem $\max\{cx : x \in P \cap X\}$ can be replaced by $\max\{cx + 0y : Fx + Gy \leq d, x \in X, y \in \mathbb{R}_+^p\}$ which is of the form (12.1) with $h = 0$.

The approach of Benders' is to decompose the problem, so as to essentially solve the problem in the x -space.

12.2 Benders' Reformulation

The main step in Benders' reformulation is to rewrite the problem (12.1) in the form

$$Z = \max_x \{cx + \phi(x) : x \in X\}, \quad (12.2)$$

where

$$\phi(x) = \max\{hy : Gy \leq d - Fx, y \in \mathbb{R}_+^p\}. \quad (12.3)$$

We assume for simplicity that $\phi(x) < \infty$ for all $x \in \mathbb{R}^n$. From Farkas' lemma or basic results concerning linear programs, see Section 2.6, this linear program defining $\phi(x)$ is feasible if and only if

$$v^t(d - Fx) \geq 0 \quad \text{for } t = 1, \dots, T$$

where $\{v^t\}_{t=1}^T$ are the extreme rays of $U = \{u \in \mathbb{R}_+^m : uG \geq h\}$.

If this linear program is feasible

$$\phi(x) = \min_{s=1, \dots, S} u^s(d - Fx)$$

where $\{u^s\}_{s=1}^S$ are the extreme points of U . This can be rewritten as $\phi(x) = \max_\eta \{\eta : \eta \leq u^s(d - Fx) \text{ for } s = 1, \dots, S\}$ leading to the reformulation of Benders', called the *Benders' Master problem*:

$$Z = \max cx + \eta \quad (12.4)$$

$$v^t(d - Fx) \geq 0 \quad \text{for } t = 1, \dots, T \quad (12.5)$$

$$u^s(d - Fx) \geq \eta \quad \text{for } s = 1, \dots, S \quad (12.6)$$

$$x \in X, \eta \in \mathbb{R}^1. \quad (12.7)$$

This is a mixed integer program with a very large (typically exponential) number of constraints. With modern mixed integer programming software, the natural way to solve such a problem is by branch-and-cut.

Note that in practice, we will often calculate $\phi(x)$ by solving the dual of the linear program (12.3), namely the the *dual separation problem*:

$$(DSP) \quad \min\{u(d - Fx) : uG \geq h, u \in \mathbb{R}_+^m\}.$$

Branch-and-Cut Algorithm for Benders'

At each node of the enumeration tree specified by the bounds $\ell \leq x \leq k$, solve the linear programming relaxation;

$$\begin{aligned} Z^* &= \max cx + \eta \\ v^t(d - Fx) &\geq 0 \text{ for } t \in T' \\ (BR) \quad u^s(d - Fx) &\geq \eta \text{ for } s \in S' \\ \ell \leq x &\leq k \\ x \in P_X, \eta &\in \mathbb{R}^1, \end{aligned}$$

where P_X is a formulation for X , i.e. $X = P_X \cap \mathbb{Z}^n$. Let \underline{Z} be the value of the incumbent.

Start with a subset $T' \subset \{1, \dots, T\}$ of the constraints (12.5) and a subset $S' \subset \{1, \dots, S\}$ of the constraints (12.6) and add to them as necessary.

Procedure.

Initialization. Place the original problem on the list of active nodes.

1. Select and remove a node from the list of active nodes.
 2. The linear programming relaxation BR is solved,
 3. If this linear program is infeasible, BR is infeasible, and the node can be pruned. Go to 1.
 4. Otherwise let (x^*, η^*) be the current linear programming solution. There are various possibilities:
 - (i) If $cx^* + \eta^* \leq \underline{Z}$, the node can be pruned by bound. Go to 1.
 - (ii) Otherwise solve DSP, the dual linear programming separation problem.
- Here there are three possible outcomes:
- (a) The dual optimal value is unbounded giving a new extreme ray v^t with $v^t(d - Gx^*) < 0$. The violated constraint

$$v^t(d - Gx) \geq 0,$$

called a *feasibility cut*, is added. Set $T' = T' \cup \{t\}$ and return to 1.

- (b) DSP is feasible, but $\phi(x^*) < \eta^*$ giving a new extreme point u^s with $\phi(x^*) = u^s(d - Gx^*) < \eta^*$. The violated constraint

$$u^s(d - Gx) \geq \eta,$$

called an *optimality cut*, is added. Set $S' = S' \cup \{s\}$ and return to 2.

(c) DSP is feasible with optimal value $\phi(x^*) = \eta^*$. All the constraints are satisfied, so the linear programming relaxation BR at the node is solved. Now there are two possible outcomes

1. x^* is integer and y^* is the corresponding solution of value $\phi(x^*)$, the incumbent value Z is updated, the solution (x^*, y^*) is stored as the best feasible solution, and the node is pruned. Go to 1.
2. x^* is not integer. Branch on one of the variables x_j taking a fractional value $(x_j^* \notin \mathbb{Z}^1)$. The (two) new nodes created are added to the active node list. Go to 1.

Example 12.1 Consider the mixed integer program

$$\begin{aligned} \max \quad & 4x_1 + 7x_2 + 2y_1 - 3y_2 + y_3 \\ & 2x_1 + 4x_2 + 4y_1 - 2y_2 + 3y_3 \leq 12 \\ & 3x_1 + 5x_2 + 2y_1 + 3y_2 - y_3 \leq 10 \\ x \quad & \leq 2, \quad x \in \mathbb{Z}_+^2, \quad y \in \mathbb{R}_+^3. \end{aligned}$$

The corresponding dual polyhedron $U = \{u \in \mathbb{R}_+^m : uG \geq h\}$ as well as its extreme points and extreme rays have been given in Example 2.4. So the complete Benders' reformulation (12.4)–(12.7) is as follows:

$$\begin{aligned} \max \quad & 4x_1 + 7x_2 + \eta \\ 12x_1 + 22x_2 \quad & \leq 56 \quad v^1 = (3, 2) \\ 11x_1 + 19x_2 \quad & \leq 42 \quad v^2 = (1, 3) \\ \frac{7}{5}x_1 + \frac{13}{5}x_2 + \eta \quad & \leq \frac{34}{5} \quad u^1 = \left(\frac{2}{5}, \frac{1}{5}\right) \\ x_1 + 2x_2 + \eta \quad & \leq 6 \quad u^2 = \left(\frac{1}{2}, 0\right) \\ 3x_1 + 6x_2 + \eta \quad & \leq 18 \quad u^3 = \left(\frac{3}{2}, 0\right) \\ x \quad & \leq 2, \quad x \in \mathbb{Z}_+^2, \quad \eta \in \mathbb{R}^1. \end{aligned}$$

Solution by the Branch-and-Cut Algorithm

Start at the initial node 1 with only the bound constraints $0 \leq x \leq 2$ and a valid upper bound on the optimal value of $\phi(x)$ (200 is taken arbitrarily).

Node 1. Iteration 1. Solve the linear programming relaxation BR with $T' = S' = \emptyset$.

$$\begin{aligned} Z^* = \max \quad & 4x_1 + 7x_2 + \eta \\ \eta \quad & \leq 200 \\ x \quad & \leq 2, \quad x \in \mathbb{R}_+^2. \end{aligned}$$

Solution of BR: $Z^* = 222$, $x^* = (2, 2)$, $\eta^* = 200$.

Solve DSP (12.8), the dual of the separation LP:

$$\begin{aligned} \max \quad & 2y_1 - 3y_2 + y_3 \\ \text{s.t.} \quad & 4y_1 - 2y_2 + 3y_3 \leq 12 - 12 \\ & 2y_1 + 3y_2 - y_3 \leq 10 - 16 \\ & y \in \mathbb{R}_+^3. \end{aligned}$$

DSP is unbounded and the extreme ray $v^2 = (1, 3)$ is found. $T' = \{2\}$. The corresponding constraint $11x_1 + 19x_2 \leq 42$ is added to BR.

Node 1. Iteration 2.

Solution of BR: $Z^* = 215 \frac{5}{11}$, $x^* = \left(\frac{4}{11}, 2\right)$, $\eta^* = 200$.

Solution of the DSP: $\phi(x^*) = \frac{12}{11} < \eta^*$. The extreme point $u^1 = \left(\frac{2}{5}, \frac{1}{5}\right)$ is dual optimal. $S' = \{1\}$.

The corresponding constraint $\frac{7}{5}x_1 + \frac{13}{5}x_2 + \eta \leq \frac{34}{5}$ is added to BR.

Node 1. Iteration 3.

Solution of BR: $Z^* = 16.6316$, $x^* = (2, 1.0526)$, $\eta^* = 1.26316$.

Solution of DSP: $\phi(x^*) = \eta^*$. BR at node 1 is solved.

The solution x^* is not integer, so we need to branch. Create node 2 by branching on $x_2 \leq 1$ and node 3 by branching on $x_2 \geq 2$, see Figure 12.1.

Node 2. Iteration 1. ($x_2 \leq 1$)

Solution of BR: $T' = \{2\}$, $S' = \{1\}$. $Z^* = 16.4$, $x^* = (2, 1)$, $\eta^* = 1.4$.

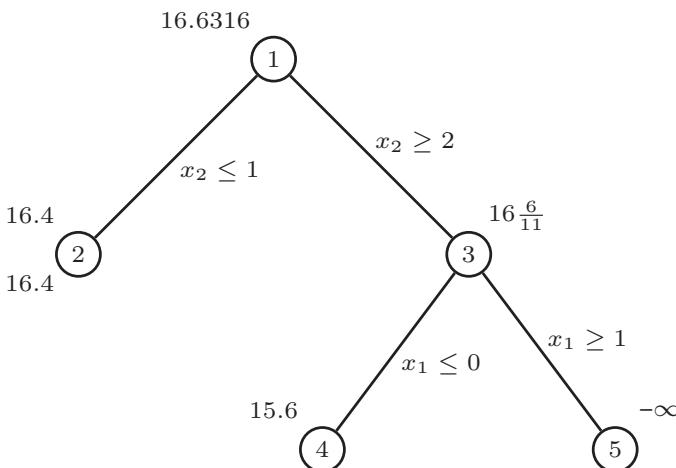


Figure 12.1 Branch-and-cut tree for Benders' algorithm.

Solution of DSP: $\phi(x^*) = \eta^*$. BR at node 2 is solved. The solution is integer. The new incumbent value $\underline{Z} = 16.4$ and the solution $x^* = (2, 1)$, $y^* = (0.1, 0, 1.2)$ is stored. The node is pruned.

Node 3. Iteration 1. ($x_2 \geq 2$)

Solution of BR: $T' = \{2\}$, $S' = \{1\}$, $Z^* = 16 \frac{6}{11}$, $x^* = \left(\frac{4}{11}, 2\right)$, $\eta^* = \frac{12}{11}$.

Solution of DSP: $\phi(x^*) = \eta^*$. The solution x^* is not integer, so we need to branch.

Create node 4 by adding $x_1 \leq 0$ and node 5 by adding $x_1 \geq 1$.

Node 4. Iteration 1. ($x_2 \geq 2, x_1 \leq 0$)

Solution of BR: $T' = \{2\}$, $S' = \{1\}$, $Z^* = 15.6$. The node is pruned by bound as $\underline{Z} = 16.4$.

Node 5. Iteration 1. ($x_2 \geq 2, x_1 \geq 1$)

Solution of BR: $T' = \{2\}$, $S' = \{1\}$

DSP: Unbounded. Extreme ray $v^1 = (3, 2)$. Feasibility cut $11x_1 + 19x_2 \leq 42$ is added to BR.

BR: $T' = \{1, 2\}$, $S' = \{1\}$. Infeasible. The node is pruned by infeasibility.

All nodes have been pruned. The search is complete. The optimal solution is $x = (1, 2)$, $y = (0.1, 0, 1.2)$ with value $Z = 16.4$. \square

12.3 Benders' with Multiple Subproblems

In many applications, the MIP has block diagonal structure of the form

$$\begin{aligned} \max cx &+ h^1 y^1 + h^2 y^2 + \dots + h^K y^K \\ F^1 x &+ G^1 y^1 \leq d^1 \\ F^2 x &+ G^2 y^2 \leq d^2 \\ \ddots &\quad \ddots \leq \vdots \\ F^K x &+ G^K y^K \leq d^K \\ x \in X, \quad y^k &\in \mathbb{R}_+^{p_k} \text{ for } k = 1, \dots, K \end{aligned}$$

Benders' reformulation can now be viewed as

$$\max \left\{ cx + \sum_{k=1}^K \phi^k(x) : x \in X \right\} \tag{12.8}$$

having K distinct separation problems

$$\eta^k \leq \phi^k(x) = \max\{h^k y^k : G^k y^k \geq d^k - F^k x, y^k \in \mathbb{R}_+^{p_k}\} \text{ for } k = 1, \dots, K$$

in place of one large separation problem. The complete Benders' reformulation is now

$$\begin{aligned} \max cx + \sum_{k=1}^K \eta^k \\ v^{t,k}(d^k - F^k x) \geq 0 \quad \text{for } t \in T^k, k = 1, \dots, K \\ u^{s,k}(d^k - F^k x) \geq \eta^k \quad \text{for } s \in S^k, k = 1, \dots, K \\ x \in X, \eta^k \in \mathbb{R}^1 \quad \text{for } k = 1, \dots, K. \end{aligned}$$

where T^k , S^k index the extreme rays and extreme points, respectively, of $\{u : uG^k \geq h^k, u \geq 0\}$ for $k = 1, \dots, K$.

We now indicate two classes of mixed integer programs having such a block diagonal structure.

1. *The Uncapacitated Facility Location Problem.* Here the reformulation takes the form:

$$\min \left\{ \sum_{j=1}^n f_j x_j + \sum_{i=1}^m \phi^i(x) : x \in \{0, 1\}^n \right\},$$

where

$$\phi^i(x) = \min \left\{ \sum_{j=1}^n c_{ij} y_{ij}, \sum_{j=1}^n y_{ij} = 1, y_{ij} - x_j \leq 0 \quad \text{for } j = 1, \dots, n, y_{i \cdot} \in \mathbb{R}_+^n \right\}.$$

2. *Two-stage Stochastic Programming with Recourse.* After taking a first stage decision $x \in X$, a random outcome (scenario) occurring with probability P_k involving one or more of the future demands d^k , the future prices e^k , and possibly the technology matrix G^k is observed. Then, an optimal second stage (recourse action) y^k depending on x and the scenario k is taken. Here $F = F^k$ and $h^k = P_k e^k$ for $k = 1, \dots, K$ and the objective function to be maximized represents the expected profit over the two stages.

$$\begin{aligned} \max \quad & cx + \sum_{k=1}^K (P_k e^k) y^k \\ & Fx + G^k y^k \leq d^k \quad \text{for } k = 1, \dots, K \\ & x \in X, \quad y^k \in \mathbb{R}_+^{P_k} \quad \text{for } k = 1, \dots, K \end{aligned}$$

Here the subproblem k takes the form:

$$\phi^k(x) = \max \{ P_k e^k y^k : G^k y^k \leq d^k - Fx, y \in \mathbb{R}_+^{P_k} \}.$$

An alternative is to group together some or all of the subproblems when sending feasibility or optimality cuts to the Master. The case in which all are grouped together leads to feasibility cuts of the form

$$\sum_{k \in K'} v^{t_k, k} (d^k - F^k x) \geq 0 \quad \text{for } K' \subseteq \{1, \dots, K\}$$

and optimality cuts of the form

$$\sum_{k=1}^K u^{s_k, k} (d^k - F^k x) \geq \eta$$

where $t_k \in T^k$ and $s_k \in S^k$ for all k .

12.4 Solving the Linear Programming Subproblems

Unfortunately the solution of the dual of the linear programming separation problem does not always provide strong cuts. Thus, when the optimum is unbounded (i.e. the primal is infeasible), the extreme ray that is found is somewhat arbitrary. Therefore, as for the lift-and-project algorithm in Section 8.9, it is usual to add a normalization constraint so that the LP has a bounded optimal value. Different normalizations such as $\sum_{i=1}^m u_i = 1$ or $\sum_{i=1}^m w_i u_i = 1$ have been suggested and tested. However, there are examples showing that the resulting cuts can be weak.

We now present three approaches that can lead to improved performance of the algorithm.

Combining the Feasibility and Optimality Criteria

Given (x^*, η^*) , a violated cut is obtained if and only if

$$\{y \in \mathbb{R}_+^p : hy \geq \eta^*, Gy \leq d - Fx^*\} = \emptyset.$$

This holds (by Farkas' lemma) if and only if $\min\{-u_0\eta^* + u(d - Fx^*) : -u_0h + uG \geq 0, (u_0, u) \in \mathbb{R}_+^1 \times \mathbb{R}_+^m\}$ is unbounded below. With the normalization $u_0 + \sum_{i=1}^m u_i = 1$, this leads to the separation problem

$$\begin{aligned} \phi(x^*) &= \min -u_0\eta^* + u(d - Fx^*) \\ &\quad -u_0h + uG \geq 0 \\ &\quad u_0 + \sum_{i=1}^m u_i = 1 \\ &\quad u_0 \in \mathbb{R}_+^1, \quad u \in \mathbb{R}_+^m. \end{aligned}$$

If $\phi(x^*) < 0$ and $u_0 = 0$ we obtain an infeasibility cut, and if $\phi(x^*) < 0$ and $u_0 > 0$ an optimality cut.

The In-Out Approach

Here the idea is to modify the point to be cutoff, typically by moving it nearer to a “central point” (η^0, x^0) that could be the best solution found so far, or some point

that is in (or close) to the feasible region. Thus, one replaces (η^*, x^*) by a point $(\hat{\eta}, \hat{x}) = \alpha(\eta^*, x^*) + (1 - \alpha)(\eta^0, x^0)$ for some $0 < \alpha < 1$.

For feasibility, the motivation is that, if \hat{x} is closer to the feasible region, then there are less inequalities $v^t(d - Gx) \geq 0$ cutting it off and thus there is a greater likelihood of generating an effective feasibility cut. For optimality cuts, the idea is similar. In the same vein, it has also been suggested to move the “central point” closer to the point (η^*, x^*) after some iterations.

Using Facet-Defining Inequalities

Consider first the case when $h = 0$. Ignoring the constraints $x \in X$, let $Q = \{(x, y) \in \mathbb{R}_+^n \times \mathbb{R}_+^p : Fx + Gy \leq d\}$ and $P = \text{proj}_x(Q)$. Then, $\max\{cx : x \in P\}$ can be reformulated as

$$\max\{cx + 0y : Fx + Gy \leq d, x \in \mathbb{R}_+^n, y \in \mathbb{R}_+^p\}.$$

This suggests that the significant infeasibility cuts $v^t(d - Fx) \geq 0$ are those corresponding to facet-defining inequalities of P .

To obtain facets of P , the following procedure can be used. Take a point $x^0 \in \text{relint}(P)$ and then make a translation $\tilde{P} = \{\tilde{x} \in \mathbb{R}^n : \tilde{x} = x - x^0, x \in P\}$ and $\tilde{Q} = \{(\tilde{x}, y) \in \mathbb{R}^m \times \mathbb{R}_+^p : F\tilde{x} + Gy \leq d - Fx^0\}$ so that now $0 \in \text{relint}(\tilde{P})$. Now the valid inequalities $\gamma\tilde{x} \leq \gamma_0$ for \tilde{P} are given by a vector $u \geq 0$ satisfying $\gamma \leq uF$, $uG \geq 0$, and $u(d - Fx^0) \leq \gamma_0$ and the point to be cutoff is $x^* - x^0$. Taking the normalization $\gamma_0 \leq 1$ gives the separation problem

$$\max \gamma(x^* - x^0) - \gamma_0 \quad (12.9)$$

$$\gamma - uF \leq 0 \quad (12.10)$$

$$-uG \leq 0 \quad (12.11)$$

$$u(d - Fx^0) - \gamma_0 \leq 0 \quad (12.12)$$

$$\gamma_0 \leq 1 \quad (12.13)$$

$$u \in \mathbb{R}_+^m. \quad (12.14)$$

It can be shown that if this LP is unbounded, the solution is an affine hyperplane containing P cutting off x^* and if the optimal value is finite and positive the solution is an inequality cutting off x^* that is almost always facet-defining for P .

This approach can be extended to the general case with $h \neq 0$ by considering $Q = \{(\eta, x, y) \in \mathbb{R}^1 \times \mathbb{R}_+^n \times \mathbb{R}_+^p : \eta \leq hy, Fx + Gy \leq d\}$ and $P = \text{proj}_{\eta, x}(Q)$.

12.5 Integer Subproblems: Basic Algorithms*

There are many problems that have the structure of (12.1) but in which all (or some) of the y variables must be integer. For simplicity, we assume that all are integer and thus $y \in \mathbb{Z}_+^p$. The y -subproblem ($\text{SP}^I(x^*)$) takes the form

$$\phi^I(x^*) = \max\{hy : Gy \leq d - Fx^*, y \in Y \subseteq \mathbb{Z}_+^p\}. \quad (12.15)$$

and therefore it is no longer just a linear program. There are now at least two possible viewpoints, either branching on both x and y variables, or just branching on the x -variables.

In this section, we present two simple algorithms. The first that involves branching in the (x, y, η) -space is a natural extension of the algorithm with $y \in \mathbb{R}_+^p$. In the second, we branch in the (x, η) -space, but the subproblems are integer programs and the cuts generated are weak.

12.5.1 Branching in the (x, η, y) -Space

The viewpoint in this subsection is:

Benders' algorithm is just a way to solve linear programs by iterating between LPs in the (x, η) -space and LPs in the y -space.

This leads to a standard branch-and-cut algorithm, except that the linear programming relaxations at each node are solved in a nonstandard way and, because of the problem structure, we choose to branch on x -variables whenever possible, only branching on y -variables when $x^* \in \mathbb{Z}_+^n$. By allowing branching on the y variables, the DSP at node $\ell \leq y \leq k$ takes the form

$$\min\{u(d - Fx^*) - u^1\ell + u^2k : (u, u^1, u^2) \in U^*\},$$

where

$$U^* = \{(u, u^1, u^2) \in \mathbb{R}_+^m \times \mathbb{R}_+^p \times \mathbb{R}_+^p : uG - u^1I_m + u^2I_m = h\}.$$

Thus, the possible set of extreme points and rays is much larger than for $U = \{u \in \mathbb{R}_+^m : uG \geq h\}$.

We return to Example 12.1 with the additional constraints $y \in \mathbb{Z}^3$.

Example 12.2 We pick up the example at node 2.

Node 2. Iteration 1.

Solution of BR: $T' = \{2\}$, $S' = \{1\}$, $Z^* = 16.4$, $x^* = (2, 1)$, $\eta^* = 1.4$.

DSP: $\phi(x^*) = \eta^*$ and $y^* = (0.1, 0, 1.2)$.

As $x^* \in \mathbb{Z}_+^2$ and $y^* \notin \mathbb{Z}^3$, we have to branch on a fractional y -variable. We choose y_3 .

Node 2a. Branch $y_3 \leq 1$ **Iteration 1.** BR: $Z^* = 16.4$, $x^* = (2, 1)$, $\eta^* = 1.4$ Subproblem has new constraint $y_3 \leq 1$.

DSP takes the form:

$$\begin{array}{lll} \min & 4u_1 - u_2 + u_3 \\ & 4u_1 + 2u_2 & \geq 2 \\ & -2u_1 + 3u_2 & \geq -3 \\ & 3u_1 - u_2 + u_3 & \geq 1 \\ & u & \geq 0. \end{array}$$

$$\phi(x^*) = 1 < \eta^*, \quad u^* = (0, 1, 2).$$

Optimality cut $\eta \leq 12 - 3x_1 - 5x_2$ added to BR.**Iteration 2.** BR. $Z^* = 16$, $x^* = (2, 1)$, $\eta^* = 1$.DSP. $\phi(x^*) = 1 = \eta^*$. $y^* = (0, 0, 1)$. BR is solved. Optimal solution is feasible. New incumbent $Z = 16$. Node is pruned.**Node 2b. Branch** $y_3 \geq 2$.**Iteration 1.** BR: $Z^* = 16.4$, $x^* = (2, 1)$, $\eta^* = 1.4$ Subproblem has new constraint $-y_3 \leq -2$.

DSP takes the form:

$$\begin{array}{lll} \min & 4u_1 - u_2 - 2u_3 \\ & 4u_1 + 2u_2 & \geq 2 \\ & -2u_1 + 3u_2 & \geq -3 \\ & 3u_1 - u_2 - u_3 & \geq 1 \\ & u & \geq 0. \end{array}$$

Dual is unbounded. Extreme ray $v = \left(\frac{3}{2}, 1, \frac{7}{2}\right)$.Infeasibility cut $6x_1 + 11x_2 \leq 21$ is added to BR.**Node 2b. Iteration 2.**BR: $Z^* = 15.6$. Node pruned by bound.

Nodes 3, 4, and 5 are unchanged from Example 12.1. Node 4 is pruned by bound and Node 5 is infeasible.

All nodes have been pruned. The search is complete. The optimal solution is $x = (1, 2)$, $y = (0, 0, 1)$ with value 16.The complete tree is shown in Figure 12.2. □

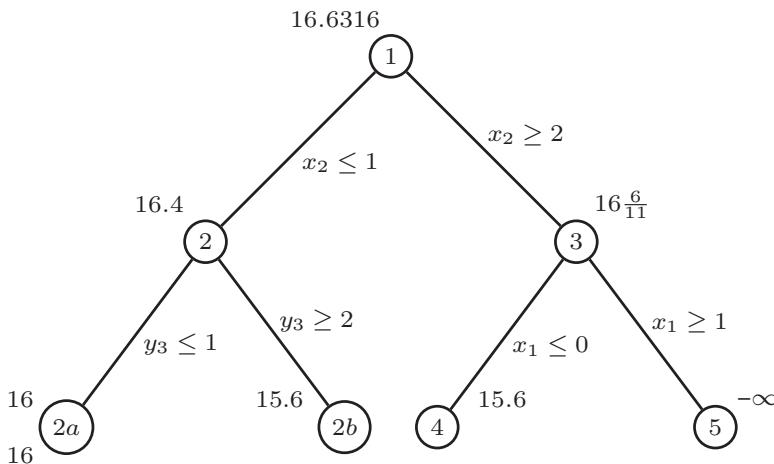


Figure 12.2 Branch-and-cut tree for Benders' algorithm: $y \in \mathbb{Z}_+^3$.

12.5.2 Branching in (x, η) -Space and “No-Good” Cuts

At a node of the branch-and-cut tree in which the linear programming solution (x^*, η^*) of BR satisfies $x^* \in \{0, 1\}^n$, a simple, but minimal approach is to solve the integer subproblem $\text{SP}^I(x^*)$ by any suitable algorithm.

- (i) If the subproblem $\text{SP}^I(x^*)$ is infeasible, x^* cannot be part of an optimal solution and the *no-good cut*

$$\sum_{j: x_j^* = 0} x_j + \sum_{j: x_j^* = 1} (1 - x_j) \geq 1 \quad (12.16)$$

is added to BR and cuts off x^* .

- (ii) If ϕ' is an upper bound on $\phi^I(x^*)$ and $\phi' < \eta^*$, the *no-good optimality cut*

$$\eta \leq \phi' + (M - \phi') \left(\sum_{j: x_j^* = 0} x_j + \sum_{j: x_j^* = 1} (1 - x_j) \right) \quad (12.17)$$

is added to BR and cuts off (η^*, x^*) where M is an upper bound on the optimal value of $\max_{x \in X} \phi^I(x)$.

- (iii) If $\phi' = \phi^I(x^*) = \eta^*$, (x^*, y^*) is a feasible solution to the original problem. If it is the best solution found so far, the incumbent is updated. The node is then pruned.

Observe that the no-good cut removes the point (x^*, η^*) , but essentially no other integer points $x \in X \setminus \{x^*\}$.

12.6 Notes

- 12.1 In the original algorithm of Benders' [41], before the existence of branch-and-cut algorithms, the relaxed Master problem in the (η, x) variables had to be solved every time that one or more feasibility or optimality cuts were added. Van Slyke and Wets [285] developed the L-shaped method which is an application of Benders' algorithm to two-stage stochastic programs. With some notable exceptions, see Geoffrion and Graves [138], there were few successful computational studies reported in the period 1960–1990. However, there were theoretical generalizations: Geoffrion [136] extended to the convex case where the subproblem was a convex program and Wolsey [297] and Caroe and Tind [64] discussed the case with integer y variables. Laporte and Louveaux published one of the first applications to stochastic integer programs, see [202]. Magnanti and Wong [218] were among the first to strengthen the Benders' cuts so that they were pareto-optimal.
- 12.2 Since around the year 2000 there has been renewed interest in the algorithm with both theory and many applications. Benders' is now best viewed as a branch-and-cut algorithm in the (x, η) -space in which the linear programming subproblem in y is a cut generating (separation) subproblem.
- 12.3 A recent survey is due to Rahamanian et al. [256]. Maher [219] describes a SCIP implementation of a framework for Benders' decomposition and Benders' algorithm is also available in Cplex.
- 12.4 The combined feasibility and optimality approach was proposed by Fischetti et al. [118]. The in-out approach was proposed in Ben-Ameur and Neto [42] and pursued by Fischetti and Salvagnin [117]. The approach with facet-defining inequalities is due to Conforti and Wolsey [75]. Recently Fischetti et al. [113, 114] have carried out tests on large-scale uncapacitated and capacitated facility location problems testing the different ideas for speeding up Benders' algorithm.
- 12.5 As many practical problems involve integer variables in the second stage, there is an important literature treating such problems. Many different ways have been suggested to deal with integrality in the subproblem. The approach of branching in the (x, y) -space is presented in Weninger and Wolsey [290]. No-good cuts have been proposed by many authors, see for instance [202]. Another approach involving branch-and-cut in the (x, η) -space involves using cutting planes for the subproblem in the y -space and extending them to obtain cuts valid in the (x, y) -space. Gade et al. [129] apply this approach using Gomory fractional cuts for a class of two level stochastic IPs and in [290] it is extended to several other classes of inequalities. For two other approaches, see [55, 255].

12.7 Exercises

1. Write an extended formulation with extreme points and extreme rays for the polyhedron

$$-x_1 + x_2 \leq 1$$

$$3x_1 + x_2 \geq 5$$

$$x_1 + x_2 \geq 1$$

$$x_1 + 2x_2 \leq 11.$$

2. Consider the mixed integer program

$$\begin{aligned} \max \quad & 4x_1 + 5x_2 + 2y_1 - 7y_2 + 5y_3 \\ & 3x_1 + 4x_2 + 2y_1 - 2y_2 + 3y_3 \leq 10 \\ & x \leq 3, x \in \mathbb{Z}_+^2, y \leq 2, y \in \mathbb{R}_+^3. \end{aligned}$$

Solve it using Benders' algorithm.

3. Consider the uncapacitated facility location problem.

Consider applying Benders' algorithm with a separable subproblem for each client i taking the form

$$Q^i = \{(x, \eta^i, y_i) : \eta^i - \sum_j c_{ij} y_{ij} \geq 0, \sum_{j=1}^n y_{ij} = 1, x_j - y_{ij} \geq 0 \forall j, x \leq \mathbf{1}, x, y_{\cdot i} \in \mathbb{R}_+^n\}.$$

Solve the separation problem by inspection.

If $c_{i1} \leq \dots \leq c_{in}$, show that

$$\begin{aligned} P^i &= \text{proj}_{x, \eta^i}(Q) \\ &= \{(x, \eta_i) : \eta_i + \sum_{j=1}^n (c_{ik} - c_{ij})^+ x_j \geq c_{ik} \text{ for } k = 1, \dots, n, \\ &\quad \sum_{j=1}^n x_j \geq 1, x \in [0, 1]^n\}. \end{aligned}$$

4. Consider the facet-generating separation LP (12.9)–(12.14) and suppose that $x^0 = 0$ and $x^* \notin P$. Write the dual of this LP. Show that it can be interpreted as
- (i) finding the smallest value $\lambda > 1$ such that $x^* \in \lambda P$, or as
 - (ii) finding the last point on the line from the origin to x^* that lies in P .
5. Suppose that $0 \leq x_j \leq 3$ for $j = 1, 2, 3$ and the point $x^* = (0, 1, 3)$ is infeasible. Derive a set of constraints cutting off the point x^* .

6. After solving the mixed integer program of Exercise 2, you are informed that the y variables should also be integer.

Without starting again from scratch

- (i) Solve the new problem using the algorithm of Section 12.5.1
- (ii) Solve using no-good cuts.

7. (i) Formulate the problem of generating a split/lift-and-project cut $\pi x \leq \pi_0$ for the Benders' Master problem BR when $x^* \notin \mathbb{Z}^n$. Apply at node 1, iteration 3 of Example 12.1.
- (ii) Formulate the problem of generating a split/lift-and-project cut $\pi x + \mu y \leq \pi_0$ for the set $W = \{(x, y) \in \mathbb{Z}_+^n \times \mathbb{R}_+^p : Fx + Gy \leq d\}$ when $x^* \notin \mathbb{Z}^n$. Apply at node 1, iteration 3 of Example 12.1.

13

Primal Heuristics

13.1 Introduction

Given that many, if not most, of the practical problems that we wish to solve are \mathcal{NP} -hard, it is not surprising that heuristic algorithms play an important role in “solving” discrete optimization problems—the idea being to hopefully find “good” feasible solutions 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 integer program (IP) or mixed integer program (MIP) of reasonable size.

Even though it has been formulated as an MIP, it is difficult or impossible for the branch-and-cut 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 MIP 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? A couple of examples of heuristics with worst-case guarantees were given in Section 6.7. Often the practical behavior of these heuristics is much better than the worst-case behavior, but is dominated in practice by some other heuristic.

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 discuss some of the ideas underlying the heuristics that have become an important component within MIP solvers in the last 15 years. Finally we suggest one or two ways in which the user can go a step further and make use of an MIP solver to develop his/her own heuristics that take into account his/her special knowledge of the problem. Often, as these approaches use an MIP solver, they provide not just a primal feasible solution but also a dual bound that provides at least some *a posteriori* performance guarantee.

13.2 Greedy and Local Search Revisited

Here we formalize the greedy and local search algorithms presented by example in Section 2.7. 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

$$\min \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \geq b, x \in \{0, 1\}^n \right\}$$

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, $v(\emptyset) = c(\emptyset) = 0$ and function v is nondecreasing. We also assume that $v(N) \geq k$ as otherwise the problem is infeasible.

A Greedy Heuristic

1. Set $S^0 = \emptyset$ (start with the empty set). Set $t = 1$.
2. Set $j_t = \arg \min_{v(S^{t-1} \cup \{j_t\}) - v(S^{t-1})} \frac{c(S^{t-1} \cup \{j_t\}) - c(S^{t-1})}{v(S^{t-1} \cup \{j_t\}) - v(S^{t-1})}$ (choose the element whose additional cost per unit of resource is minimum).
3. If the previous solution S^{t-1} is feasible, i.e. $v(S^{t-1}) \geq k$, and $c(S^{t-1} \cup \{j_t\}) \geq c(S^{t-1})$, stop with $S^G = S^{t-1}$.
4. Otherwise set $S^t = S^{t-1} \cup \{j_t\}$.
5. If $t = n$, stop with $S^G = N$.
6. Otherwise set $t \leftarrow t + 1$, and return to 2.

Example 13.1 We apply the greedy heuristic to an instance of the uncapacitated facility location problem with $m = 6$ clients, $n = 5$ depots, and costs

$$(c_{ij}) = \begin{pmatrix} 6 & 9 & 3 & 4 & 12 \\ 1 & 2 & 4 & 9 & 2 \\ 15 & 2 & 6 & 3 & 18 \\ 9 & 23 & 4 & 8 & 1 \\ 7 & 11 & 2 & 5 & 14 \\ 4 & 3 & 10 & 11 & 3 \end{pmatrix} \quad \text{and } f = (21, 16, 30, 24, 11).$$

As $v(S) = |S|$, $v(S \cup \{j\}) - v(S) = 1$ for all $j \notin S$. The algorithm gives:

Initialization. $S^0 = \emptyset$. S^0 is infeasible.

Iteration 1. $c(1) = (6 + 1 + 15 + 9 + 7 + 4) + 21 = 63$, $c(2) = 66$, $c(3) = 59$, $c(4) = 64$, $c(5) = 61$, so $j_1 = 3$, $S^1 = \{3\}$ and $c(S^1) = 59$. S^1 is feasible.

Iteration 2. $c(1, 3) - c(3) = (\min\{6, 3\} + \min\{1, 4\} + \min\{15, 6\} + \min\{9, 4\} + \min\{7, 2\} + \min\{4, 10\}) + (21 + 30) - 59 = 12$, $c(2, 3) - c(3) = 3$, $c(3, 4) - c(3) = 21$, $c(3, 5) - c(3) = -1$. $S^2 = \{3, 5\}$ and $c(S^2) = 58$. S^2 is feasible.

Iteration 3. $c(3, 5, j) - c(3, 5) > 0$ for $j \in \{1, 2, 4\}$.

The greedy solution is $S^G = \{3, 5\}$ of cost 58. \square

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 $g(S) \geq 0$ 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* $Q(S)$ 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 13.1). This leads to the well-known 2-exchange heuristic for the STSP on a complete graph.

2-Exchange Heuristic for STSP

The local search heuristic is applied with the following specifications:

A set $S \subset E$ is a solution if the set of edges S form a tour.

$$Q(S) = \{S' \text{ is a solution: } S' \neq S, |S' \cap S| = n - 2\}, \text{ where } n = |V|.$$

$$f(S) = \sum_{e \in S} c_e.$$

The resulting local search solution is called a *2-optimal tour*.

GRASP (Greedy Randomized Adaptive Search Procedure) is a way to combine greedy and local search and at the same time generate a variety of heuristic solutions. The basic idea is just to modify the greedy algorithm. Rather than adding the greedy or “best” choice of item at each step, a random choice is made among the k -best available items. Then after a randomized greedy solution has been obtained, the solution becomes the input to an appropriate local search heuristic.

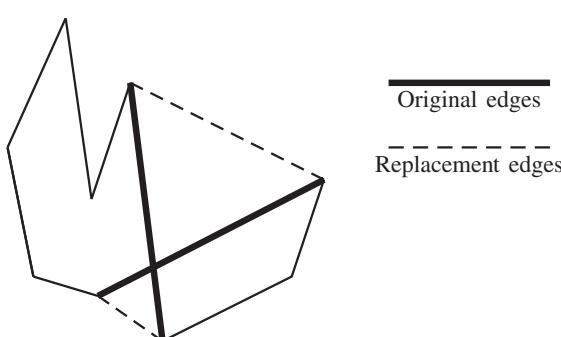


Figure 13.1 2-Exchange
for STSP.

13.3 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.

13.3.1 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 as follows:

- (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 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.

13.3.2 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, it is necessary 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 2.7). 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 13.3.1. 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.

13.3.3 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 $x_1x_2 \cdots x_r$.

Three possible ways to combine such strings are as follows:

1-Point Crossover. Given two strings $x_1x_2 \cdots x_r$ and $y_1y_2 \cdots y_r$, and an integer $p \in \{1, \dots, r-1\}$, the two children are $x_1 \cdots x_p y_{p+1} \cdots y_r$ and $y_1 \cdots y_p x_{p+1} \cdots x_r$.

2-Point Crossover. Given two strings $x_1x_2 \cdots x_r$ and $y_1y_2 \cdots y_r$, and integers $p, q \in \{1, \dots, r-1\}$ with $p < q$, the two children are $x_1 \cdots x_p y_{p+1} \cdots y_q x_{q+1} \cdots x_r$ and $y_1 \cdots y_p x_{p+1} \cdots x_q y_{q+1} \cdots y_r$.

Uniform Crossover. Given two strings $x_1x_2 \cdots x_r$ and $y_1y_2 \cdots y_r$, the result is a child $z_1 \cdots z_r$ 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 $z_1 \cdots z_r$ is a random choice of $p \in \{1, \dots, r\}$ and a random integer \tilde{z}_p from the appropriate range giving a modified solution $z_1 \cdots z_{p-1} \tilde{z}_p z_{p+1} \cdots z_r$. 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:

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} \quad (13.1)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for } i \in M \quad (13.2)$$

$$\sum_{i \in M} a_{ij} x_{ij} \leq b_j \quad \text{for } j \in N \quad (13.3)$$

$$x \in \{0, 1\}^{|M| \times |N|}. \quad (13.4)$$

S is a “solution” if its incidence vector satisfies (13.2) and (13.4). 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 $g(S) = \sum_{j \in N} \left(\sum_{i \in M} a_{ij} - b_j \right)^+$.

The two-point crossover and the two-point mutation described above, both lead to new solutions satisfying (13.2) and (13.4).

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.

13.4 Heuristics Inside MIP Solvers

For simplicity most heuristics are described for the IP: $\max\{cx : x \in X \subset \mathbb{Z}_+^n\}$, but are readily generalized to MIPs: $\max\{cx : x \in X \subseteq \mathbb{Z}^n \times \mathbb{R}^p\}$.

13.4.1 Construction Heuristics

First, we consider construction heuristics whose goal is to build a feasible solution from scratch. Depending on the time available, there are heuristics that just use elementary operations, others that solve linear programs and finally others that even solve MIPs of reduced size. Below we list some of the ideas used and then discuss in a little more detail a couple of well-known heuristics.

Starting from an arbitrary point, the origin or some known point in the feasible LP region P ,

- (i) *Rounding*. Select a variable x_j that should be integer and round to the nearest integer, denoted $\lceil x_j \rceil$.
 - (ii) *Shift*. Shift x_j to $x_j \pm \alpha$ if this decreases infeasibility.
 - (iii) *Fix*. Set x_j to some value $x'_j \in \mathbb{Z}^1$ (using preprocessing or constraint programming to tighten bounds due to the fixing).
- If repetition of these steps does not lead to a feasible solution,
- (iv) Solve a linear program with the variables that have been fixed and round the solution.
- If this still does not lead to a feasible solution:

- (v) After possibly fixing more variables based on the previous step, solve a mixed integer program for a limited amount of time. Each of these steps is possibly repeated many times.

We now describe three of the resulting heuristics, all of which use LP. The first two are very simple. The third, called the feasibility pump, is more interesting.

A Dive-and-Fix Heuristic

We suppose that we have a 0–1 problem. Given an LP solution x^* , let $F = \{j : x_j^* \notin \{0, 1\}\}$ be the set of 0–1 variables that are fractional.

Initialization. Take the LP solution x^* at some node.

Basic Iteration. As long as $F \neq \emptyset$,

Let $i = \arg \min_{j \in F} \{\min[x_j^*, 1 - x_j^*]\}$ (find the variable closest to integer).

If $x_i^* < 0.5$, fix $x_i = 0$ (if close to 0, fix to 0).

Otherwise set $x_i = 1$ (if close to 1, fix to 1).

Solve the resulting LP.

If the LP is infeasible, stop (the heuristic has failed).

Otherwise let x^* be the new LP solution.

Termination. If $F = \emptyset$, x^* is a feasible mixed integer solution.

The Neighborhood Rounding Heuristic

Given an optimal solution of the LP relaxation x^* , solve the IP

$$\max\{cx : x \in X, \lfloor x_j^* \rfloor \leq x_j \leq \lceil x_j^* \rceil \text{ for } j \in N\}.$$

The Feasibility Pump Heuristic

Initialization. Let $x^* = x^{LP}$ be an optimal LP solution.

Iteration. Round to obtain an integer point $\hat{x} = \lceil x^* \rceil$.

If $\hat{x} \notin P$, project onto the feasible region P .

If the IP is in the form $\max\{cx : Ax \leq b, \ell \leq x \leq k, x \in \mathbb{Z}^n\}$, the projection problem: $\min\{\sum_j |x_j - \hat{x}_j| : x \in P\}$ can be formulated as the linear program:

$$\begin{aligned} \min \quad & \sum_{j \in B: \hat{x}_j = \ell_j} (x_j - \ell_j) + \sum_{\in B: \hat{x}_j = k_j} (k_j - x_j) + \sum_{j: \ell_j < \hat{x}_j < k_j} \delta_j \\ \text{Ax} \leq b \\ \delta & \geq x - \hat{x} \\ \delta & \geq \hat{x} - x \\ \ell & \leq x \leq k \end{aligned}$$

where variables $\delta_j = |x_j - \hat{x}_j|$ for all j are used to model the integer variables lying between their bounds. Let z be an optimal solution.

Set $x^* = z$ and repeat.

In many cases, the basic version of the heuristic is modified so as to more explicitly take into account the objective function, and thus hopefully find a better feasible solution. For example, some decreasing weighting of the real objective function cx is added to the objective function of the LP in the projection step during the initial iterations. Also rather than just rounding the LP solution x^* , all the points close to a line joining x^* to a “central” or interior point x^0 can be rounded to obtain an integer point that is closest to satisfying the constraints $Ax \leq b, \ell \leq x \leq u$.

13.4.2 Improvement Heuristics

Given a feasible solution $x^* \in X$, several interesting ideas have been proposed for finding improved solutions. Nearly all of them involve the solution of an MIP in which many variables are fixed or have restricted ranges so that the MIP has a good chance of finding a better feasible solution quickly.

A Local Branching Heuristic

Given $x^* \in X$, solve an IP in which the feasible solutions lie in a small neighborhood of x^* . Specifically if $X \subseteq \{0, 1\}^n$, solve the IP:

$$\max \left\{ cx : x \in X \cap \left\{ x : \sum_{j:x_j^*=0} x_j + \sum_{j:x_j^*=1} (1-x_j) \leq k \right\} \right\},$$

where k is a small integer.

Reversing the roles of the local branching constraint and the objective function leads to a variant.

A Proximity Search Heuristic

Again we suppose that $x^* \in X \subset \{0, 1\}^n$ is given. Selecting an improvement parameter $\delta > 0$, the idea is to find a feasible solution that improves on the best available solution by at least δ and is as close as possible to the starting solution x^* . This leads to the IP:

$$\begin{aligned} \min(\sum_{j:x_j^*=0} x_j + \sum_{j:x_j^*=1} (1-x_j)) \\ \sum_{j=1}^n c_j x_j \leq cx^* - \delta \\ x \in X. \end{aligned}$$

A Relaxation Induced Neighborhood Search (RINS) Heuristic

Given a feasible solution $x^* \in X$ and an optimal LP solution x^{LP} , let $F = \{j \in N : x_j^* = x_j^{LP}\}$. Solve the IP:

$$\max\{cx : x \in X \cap \{x : x_j = x_j^* \text{ for } j \in F\}\}.$$

A Polishing Heuristic

Motivated by genetic algorithms, a list of at most k best solutions is stored. Two or more solutions $\{x^1, \dots, x^r\}$ are selected and then the idea is to fix the set $F = \{j \in [1, n] : x_j^1 = x_j^t \text{ for } t = 1, \dots, r\}$. Then again the IP

$$\max\{cx : x \in X \cap \{x : x_j = x_j^* \text{ for } j \in F\}\}$$

is solved. Here the selection of a pair of solutions is obtained by choosing a first solution randomly and then randomly choosing a second solution among the better solutions on the list. Mutations are introduced by randomly fixing only a subset of the variables in F .

Several of these heuristics have been incorporated in the recent versions of the MIP solvers. There are naturally minor variants and typically the size of the fixed set F must be adjusted so that the resulting MIP solves quickly, but also produces feasible solutions.

13.5 User-Defined MIP heuristics

Given an MIP, there are many cases in which the user, making use of his knowledge of an instance or its problem class, can either adapt one of the heuristics cited above, or develop his own specific heuristic. Below we present two such heuristics, one a construction and the second an improvement heuristic that have proven useful in several cases in which the MIP codes fail to find very good solutions.

Relax-and-Fix Heuristic

Here we suppose that there is a partition of the variables and the problem can be written in the form:

$$Z = \max\{c^1x^1 + c^2x^2 : A^1x^1 + A^2x^2 = b, x^1 \in \mathbb{Z}_+^{n_1}, x^2 \in \mathbb{Z}_+^{n_2}\}.$$

The steps of the heuristic are then:

1. *Relax.* Solve the relaxation

$$\begin{array}{lll} \bar{Z} = \max & c^1x^1 + c^2x^2 \\ (\text{MIP1}) & A^1x^1 + A^2x^2 = b \\ & x^1 \in \mathbb{Z}_+^{n_1}, x^2 \in \mathbb{R}_+^{n_2} \end{array}$$

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

$$\begin{array}{ll} \underline{Z} = \max c^1 x^1 + c^2 x^2 \\ (\text{MIP2}) \quad \begin{array}{rcl} A^1 x^1 + A^2 x^2 & = & b \\ x^1 & = & \bar{x}^1 \\ x^2 & \in & \mathbb{Z}_+^{n_2} \end{array} \end{array}$$

Let (\bar{x}^1, \tilde{x}^2) be the corresponding solution if MIP2 is feasible.

3. *Heuristic Solution.* The heuristic solution is $x^H = (\bar{x}^1, \tilde{x}^2)$ with $\underline{Z} = cx^H \leq Z \leq \bar{Z}$.

In practice, we often extend this idea by breaking up the set of variables into more than two subsets. For instance given a problem involving T time periods, we might break up the interval into K subintervals $[t_k, \tau_k]$ with $t_1 = 1$, $\tau_K = T$, and $t_{k+1} = \tau_k + 1$ for $k = 1, \dots, K - 1$ leading to a corresponding partition of the variables.

A more general version is obtained by working with triples $[t_k, \sigma_k, \tau_k]$ with $t_k \leq \sigma_k \leq \tau_k$, $t_k = \sigma_{k-1} + 1$. In the k th step, we solves the MIP

$$\max\{cx : x \in X, x_j = x_j^F \text{ for } j = 1, \dots, \sigma_{k-1},$$

$$x_j \in \mathbb{Z}_+^1 \text{ for } j = t_k, \dots, \tau_k, x_j \in \mathbb{R}_+^1 \text{ for } j > \tau_k\}$$

with optimal solution x^* and then set $x_j^F = x_j^*$ for $j = t_k, \dots, \sigma_k$.

Thus, for an instance with 30 periods in which we allow 10 variables to be integer in each restricted MIP and then fix the values of the first 6 integer variables, the corresponding values for the 5 MIPs are as follows: [1, 6, 10], [7, 12, 16], [13, 18, 22], [19, 24, 28], and [25, 30, 30].

Large Neighborhood Search Heuristic

This is a local search heuristic. To optimize over a large neighborhood either MIP or a specialized algorithm is used. Typically, the MIP is run until an improved solution is found or a time limit is reached. Given the problem in the form $\max\{cx : x \in X\}$ and a feasible solution $x^* \in X$, the basic idea is to define a subset of variables $V \subset N$ whose values it is hoped to improve by solving the MIP:

$$\max\{cx : x \in X, x_j = x_j^* \text{ for } j \in N \setminus V\}.$$

For example after finding a feasible solution to the 30 period instance by Relax-and-Fix, one possibility is to set $V = [7, 12]$ to see if the solution between periods 7 and 12 can be improved. Alternatively, if the instance involved five items, we might fix the solution of items 1–4 and see if the solution of item 5 could

be improved. Note that the Local Branching, relaxation induced neighborhood search (RINS), and Proximity Search heuristics are also of this type.

Extended Formulation Heuristics

Several problems can be formulated either in the original variables $\max\{cx : x \in P \cap \mathbb{Z}^n\}$ or with an extended formulation $\max\{cx + 0w : (x, w) \in Q \cap (\mathbb{Z}^n \times \mathbb{R}^p)\}$ where the first formulation provides very weak dual bounds, whereas the second provides good dual bounds, but is so large that it is only practical to solve its LP relaxation.

Here, the first option is to reduce the size of the extended formulation. Use the optimal LP solution (x^{LP}, w^{LP}) to fix sufficient variables $F_x \subseteq \{j \in [1, n] : x_j^{LP} \in \mathbb{Z}_+^1\}$ so that the restricted MIP

$$\max\{cx + 0w : x_j = x_j^{LP} \text{ for } j \in F_x, (x, w) \in Q \cap (\mathbb{Z}^n \times \mathbb{R}^p)\}$$

can be run to produce hopefully good feasible solutions.

A second option is to fix sufficient variables F_x so that the restricted original formulation

$$\max\{cx : x_j = x_j^{LP} \text{ for } j \in F_x, x \in P \cap \mathbb{Z}^n\}$$

produces good quality feasible solutions.

Problem Specific Heuristics

To give an example, Production Routing problems involve decisions on what to produce and when, and also the choice of when and how much each vehicle should deliver to a customer. MIP unfortunately does not deal effectively with the routing aspects and often gives primal and dual bounds that are far from optimal.

One idea that has proven effective is to decompose the problem. First, the complete problem is simplified by dropping the routing aspect and replacing it by a fixed cost for visiting a client in each period. This production-assignment problem can typically be solved to optimality. It provides details on which clients are visited in each period and the quantity delivered to them. The remaining problem is a vehicle routing problem or a set of TSPs for which very good heuristics are available. The idea is then to use the vehicle routes provided by the second heuristic to update the client visiting costs. Specifically, if client i lies on the route of vehicle v in period t , her/his assignment cost is set to be the marginal cost of inserting i into the route, namely $C_{ivt} = c_{i-i} + c_{ii^+} - c_{i^-i^+}$, where i^-, i^+ are the predecessor and successor of client i on route v . On the other hand, if the client i does not lie on route v , C_{ivt} is set to be the cost of the cheapest insertion into route v . Then, iterating

between the two problems, the aim is to arrive at a production assignment whose real routing cost has been “well” approximated.

13.6 Notes

The book of Aarts and Lenstra [1] was 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* Fisher and Rinnooy Kan [121] was dedicated to the subject of heuristics. Since then there has been an explosion of articles on heuristics as well as a specialized journal, the Journal of Heuristics.

- 13.2 Greedy and local search heuristics are part of the folklore. GRASP was proposed in Feo and Resende [109]. A very effective generalization of the 2 and 3-exchange heuristics was proposed by Lin and Kernighan [208].
- 13.3 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 Glover [146–148]. The origins of simulated annealing heuristics are attributed to Metropolis et al. [227], Kirkpatrick et al. [197], and Cerny [66]. 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 bibliography of simulated annealing and tabu search is Osman and Laporte [236]. Genetic algorithms originated with the work of Holland [180] and Goldberg [151].
Other heuristic approaches include ant colony heuristics, see Dorigo and Stützle [100]. Constraint integer programming, see Achterberg [3], provides an alternative approach for certain discrete problems.
- 13.4 Shifting and propagating are explained in Bertold and Hendel [46]. Variants of the dive-and-fix heuristic are common. For the feasibility pump heuristic, see Fischetti et al. [112], Achterberg and Bertold [4], and Fischetti and Salvagnin [116]. The local search heuristic was proposed in Fischetti and Lodi [115], the RINS heuristic in Dana et al. [89], and polishing in Rothberg [259].
- 13.5 The relax-and-fix heuristic is also well known. Two examples of the relax-and-fix heuristics are in Bienstock and Günlük [52] and Toledo et al. [275], respectively. There are many variants of Local Search heuristics, see

among others Hendel [177]. The use of approximate routing costs for routing can be found in Fisher and Jaikumar [120]. Their use in a production routing problem appears in Absi et al. [2].

13.7 Exercises

1. Devise a simple heuristic for the instance of GAP in Exercise 4 in Chapter 10. Then apply your heuristic to a larger instance and modify it, if necessary.
2. Consider the generalized transportation problem:

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

- (i) Propose a heuristic for this problem.
(ii) Propose an exact algorithm.
3. Apply the different heuristics presented to an instance of STSP with the following distance matrix:

$$\left(\begin{array}{cccccc} - & & & & & \\ 28 & - & & & & \\ 57 & 28 & - & & & \\ 72 & 45 & 20 & - & & \\ 81 & 54 & 3 & 10 & - & \\ 85 & 57 & 28 & 20 & 22 & - \\ 80 & 63 & 57 & 72 & 81 & 63 & - \end{array} \right).$$

Devise at least one new heuristic and apply it to this instance.

4. Apply greedy and local neighborhood heuristics to an instance of the problem of most profitably allocating clients to at most K depots:

$$\begin{aligned} \max & \sum_{i \in M} \sum_{j \in N} c_{ij} y_{ij} \\ & \sum_{j \in N} y_{ij} = 1 \quad \text{for } i \in M \\ & y_{ij} - x_j \leq 0 \quad \text{for } i \in M, j \in N \\ & \sum_{j \in N} x_j \leq K \end{aligned}$$

$$y \in \{0, 1\}^{mn}, x \in \{0, 1\}^n,$$

with $m = 7$ clients, $n = 6$ potential depots, $K = 3$, and

$$(c_{ij}) = \begin{pmatrix} 2 & 3 & 7 & 3 & 6 & 1 \\ 3 & 1 & 1 & 8 & 10 & 4 \\ 6 & 2 & 3 & 1 & 2 & 7 \\ 8 & 1 & 4 & 6 & 2 & 3 \\ 4 & 4 & 3 & 3 & 4 & 3 \\ 2 & 8 & 3 & 6 & 3 & 2 \\ 6 & 5 & 3 & 2 & 7 & 4 \end{pmatrix}.$$

14

From Theory to Solutions

14.1 Introduction

The aim in this final chapter is to indicate how the ideas presented earlier can be put to use to improve the formulation and solution of integer programs. Specifically, we discuss briefly the type of software available, then we ask *how* an integer programmer might look for a new valid inequality, or an improved formulation for a specific problem, and finally we look at two practical applications and ask *how* an integer programmer might try to solve them. The goal is not to study any problem exhaustively, nor to present computational results for different approaches, but rather to indicate the questions to be asked and the steps to be considered in trying to produce results for a particular problem. We terminate with a very brief discussion of some recent links between machine learning and integer programming that will undoubtedly increase in the future.

14.2 Software for Solving Integer Programs

Two essential tools for an integer programmer are a modeling or programming language and a mixed integer programming system.

A *modeling language* provides a way for the user to write out a *model* of his mixed integer programming formulation in a formal way that closely resembles the mathematical formulations used throughout this book. In addition, such languages allow recuperation of the data from files or databases, and calculations based on the data. The output of the modeling language is typically a representation of the MIP instance in a standard (but far from readable) “MPS” format or an “LP” format in which the objective function, constraints, and bounds are written out explicitly.

There are at least two advantages in using a modeling language. First, the model provides documentation, making it relatively easy to return to a model several weeks or months later. Second, with such a model, changes to a formulation can be made very easily and quickly. The modified model can often be resolved within seconds or minutes.

An example of a model in a representative language is shown below.¹ This formulation and instance are from Exercise 10 in Chapter 3.

```
!_____
model FNCF !Fixed Charge Network Flow Model
    !!Modeling Language MOSEL
options noimplicit
uses "mmxprs"
declarations
N: integer      !Number of Nodes
M: integer      !Number of Arcs
end-declarations

N:=4
M:=8

declarations
B: array(1..N) of real
TAIL: array(1..M) of integer !Tail of arc e
HEAD: array(1..M) of integer !Head of arc e
C: array(1..M) of real ! Fixed cost of arc e
x: array(1..M) of mpvar ! 0-1 variable if arc e is open
y: array(1..M) of mpvar ! Flow in arc e
BAL: array(1..N) of linctr ! Balance constraints at each node
VUB: array(1..M) of linctr ! Variable upper bound constraint on
each arc
OBJ: linctr ! The objective function
end-declarations
!_____
! Data usually read from files
B:=[-6,2,3,1]
TAIL:=[1,1,4,1,2,3,4,3]
HEAD:=[4,2,2,3,3,2,3,4]
C:=[5,2,7,3,2,4,9,12]
!_____
!Constructing the model
!CONSTRAINTS
! Flow conservation at node i
forall(i in 1..N)
BAL(i):= sum(e in 1..M|HEAD(e)=i)y(e)-sum(e in 1..M|TAIL(e)=i)y(e)
=B(i)
```

¹ The example, written in the FICO® Xpress Mosel modeling and programming language, is used with permission from Fair Isaac Corporation. FICO is a trademark of Fair Isaac Corporation.

```

! Variable upper bound constraint on arc e
forall(e in 1..M) do
VUB(e):= y(e)<= -B(1)*x(e)
x(e) is_binary
end-do

OBJ:=sum(e in 1..M)C(e)*x(e)
!_____
! Solving the model
setparam("XPRS_VERBOSE",1)
minimize(OBJ)

! Writing out the optimal solution
forall(e in 1..M| getsol(y(e))>0)do
writeln("flow on arc ",e,"= (",TAIL(e),",",HEAD(e),") is ",
getsol(y(e)))
writeln("xval ",e,"= ",getsol(x(e)))
end-do

end-model
!_____

```

A *mixed integer programming system* is a program that reads in a mixed integer problem instance either in one of these special formats, or through an interface written in C, C++, Python, Java, etc. depending on the solver, or directly from a modeling language. It creates an internal representation of the instance and solves it using branch-and-cut. Most modern systems include presolve, heuristics such as the feasibility pump, RINS, and local branching, and cutting plane generation including lifted knapsack cover inequalities, mixed integer rounding inequalities, and Gomory mixed integer cuts among others. These systems can often solve instances with tens or hundreds of thousands of variables. On the other hand, there are classes of problems with relatively few integer variables that cannot be solved even after several hours. These require either better formulations, specialized heuristics, problem specific cutting planes or possibly the use of a decomposition algorithm.

The MIP systems also provide subroutine optimization libraries with callbacks accessed through an appropriate programming language interface that enable the user to build his own special purpose algorithm by including a new class of cuts, or a new branching strategy or variable selection rule, or a problem-specific heuristic. Some of the MIP systems have now been extended to handle certain nonlinearities and/or other features and algorithms such as

- (i) convex quadratic objectives and constraints
- (ii) nonconvex quadratic constraints
- (iii) second-order cone programming
- (iv) linearization of nonlinear separable terms such as $\sum_{j=1}^n f(x_j)$

- (v) constraint programming
- (vi) Benders' algorithm
- (vii) branch-cut-and-price.

14.3 How Do We Find an Improved Formulation?

We first examine two simple lot-sizing models that appear as submodels in the production planning application to be examined in Section 14.4. The two models, both of which have been discussed earlier, differ in their complexity.

Uncapacitated Lot-Sizing

Consider the set $X^{\text{ULS}} =$

$$\left\{ (x, s, y) \in \{0, 1\}^n \times \mathbb{R}_+^n \times \mathbb{R}_+^n : \begin{array}{l} s_{t-1} + y_t \\ = d_t + s_t, y_t \leq \left(\sum_{i=t}^n d_i \right) x_t \quad \text{for } t = 1, \dots, n \end{array} \right\}$$

We typically ask a series of questions.

Q1. Is optimization over X^{ULS} polynomially solvable?

A1. Yes. This is shown in Section 5.2. Therefore, as indicated in Chapters 2 and 6, there is some hope of finding an explicit description of $\text{conv}(X^{\text{ULS}})$.

One way to improve a formulation is to add valid inequalities. To make a start at finding a valid inequality, one approach is to look at the nonintegral solutions obtained when solving the linear programming relaxation of $\min\{fx + hs + py : (x, s, y) \in X^{\text{ULS}}\}$.

We consider an instance with $d = (6, 7, 4, 6, 3, 8)$, $p = 0$, $h = (1, 1, 3, 1, 1, 2)$, and $f = (8, 12, 8, 6, 10, 23)$. As $d_1 > 0$, we immediately add $x_1 = 1$ to the formulation.

The resulting linear programming solution (x^1, s^1, y^1) is shown in Figure 14.1.

Q2. How can the point (x^1, s^1, y^1) be cut off?

Observation 14.1 As in the development of the dynamic programming recursion for ULS, the solution decomposes between successive points with $s_t = 0$. So it suffices to look at the solution lying between $s_1^1 = 0$ and $s_2^1 = 0$, namely $x_2^1 = 0.25$, $y_2^1 = 7$, $s_1^1 = 0$, $s_2^1 = 0$ for which the x variable is not integer.

A2a. Try to use known cutting planes. Consider the set

$$\left\{ (x_2, y_2, s_1, s_2) \in \{0, 1\}^1 \times \mathbb{R}_+^1 \times \mathbb{R}_+^2 : s_1 + y_2 = d_2 + s_2, y_2 \leq \left(\sum_{i=2}^n d_i \right) x_2 \right\}.$$

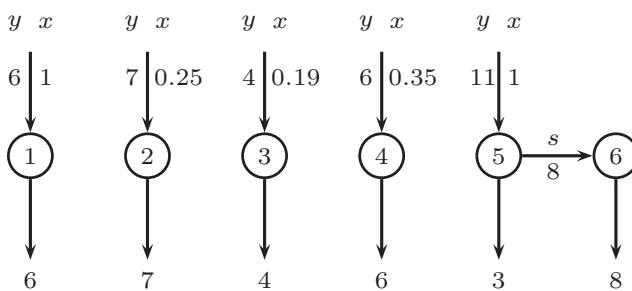


Figure 14.1 First ULS solution.

The flow cover inequality with variable y_2 in the cover C and excess $\lambda = \sum_{i=2}^n d_i - d_2$ gives the inequality $y_2 + d_2(1 - x_2) \leq d_2 + s_2$, or in general

$$y_t \leq d_t x_t + s_t \quad \text{for } t = 1, \dots, n.$$

- A2b. Try to find a valid inequality directly. Again just consider the partial solution $x_2^1 = 0.25$, $y_2^1 = 7$, $s_1^1 = 0$, $s_2^1 = 0$. Logically in any feasible solution to ULS with $x_2 = 0$, there is no production in period 2, and so the demand $d_2 = 7$ for period 2 must be contained in the entering stock s_1 . Thus, $s_1 \geq 7$ if $y_2 = 0$. Now we try to convert this observation into a valid inequality. The inequality $s_1 \geq d_2(1 - x_2)$ does the trick when $x_2 = 0$. So we just need to check that it is also valid when $x_2 = 1$. But in this case, the inequality reduces to $s_1 \geq 0$, which is a constraint of the initial problem. So in general we have a valid inequality

$$s_{t-1} \geq d_t(1 - x_t) \quad \text{for } t = 1, \dots, n.$$

Using the equality $s_{t-1} + y_t = d_t + s_t$, it is readily checked that the two inequalities we have come up with are the same, and those for $t = 2, 3, 4$ cut off the linear programming solution (x^1, s^1, y^1) .

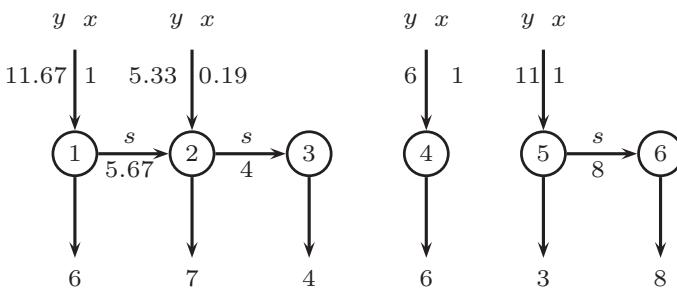
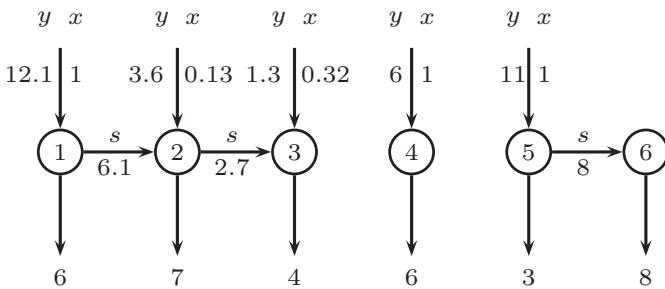
To go a step further, we add these inequalities to the formulation, and the new linear programming solution is as shown in Figure 14.2.

Again the question is how to cut off the point (x^2, s^2, y^2) . As $s_3^2 = 0$, it suffices to look at the non-integral solution for periods 1–3.

- A2c. Extending the argument in A2b, we observe that if there is no production in periods 2 and 3, then $s_1 \geq d_2 + d_3 = 11$. This immediately gives the family of valid inequalities

$$s_{k-1} \geq \left(\sum_{t=k}^l d_t \right) (1 - x_k - \dots - x_l) \quad \text{for } 1 \leq k \leq l \leq n$$

and among them $s_1 \geq 11(1 - x_2 - x_3)$ cuts off (x^2, s^2, y^2) .

**Figure 14.2** Second ULS solution.**Figure 14.3** Third ULS solution.

Again we add these inequalities to the formulation, and the new linear programming solution is as shown in Figure 14.3.

Here we observe that the cuts added in the last two iterations are both satisfied at equality. Somehow we need something stronger.

- A2d. We have used the two valid inequalities $s_1 \geq d_2(1 - x_2)$ and $s_1 \geq (d_2 + d_3)(1 - x_2 - x_3)$. Comparing the two, we see that the d_2 term in the second inequality is weaker than that in the first. The valid inequality

$$s_1 \geq d_2(1 - x_2) + d_3(1 - x_2 - x_3)$$

takes this observation into account, and has exactly the right condition needed to include the demands d_2 , or $d_2 + d_3$. After adding this inequality, we obtain a solution with x integer. The general form of this last inequality is

$$s_{k-1} \geq \sum_{t=k}^l d_t(1 - x_k - \dots - x_t) \quad \text{for } 1 \leq k \leq l \leq n. \quad (14.1)$$

Before leaving this example, there are still several questions that we might ask about the use and strength of the inequalities we have found.

- Q3. Are the inequalities (14.1) as strong as possible (facet-defining), or are they nonnegative combinations of other valid inequalities?
- A3. Yes. The inequalities are facet-defining, see Exercise 5.
- Q4. Do the constraints of the original formulation plus the $O(n^2)$ inequalities (14.1) describe $\text{conv}(X^{\text{ULS}})$?
- A4. No. The inequality $y_k \leq (\sum_{t=k}^l d_t)x_k + s_l$ is valid and facet-defining for any $l \geq k$, and is not equivalent to any inequality of the form (14.1) when $l > k$. However, the inequalities (14.1) are sufficient to solve many practical instances.
- Q5. Is the separation problem for the inequalities (14.1) and a point (x^*, s^*, y^*) easy?
- A5. Yes. Fix k , and find the smallest value $l \geq k$ for which $x_k^* + \dots + x_l^* \geq 1$. If $l > k$, check if

$$s_{k-1}^* < \sum_{t=k}^{l-1} d_t(1 - x_k^* - \dots - x_t^*).$$

If so, this is the most violated inequality for the given value of k . If not, no such inequality is violated.

In conclusion, there is the option of either reformulating by adding the $O(n^2)$ inequalities a priori, or of using a simple separation routine to generate the inequalities (14.1) as cuts when they are violated.

- Q6. Is there an extended formulation for ULS?
- A6. There are several including the one presented in Section 1.6. In spite of their strength, these formulations typically have the disadvantage of having an order of magnitude more variables and constraints.

Capacitated Lot-Sizing

Here we consider the more constrained set $X^{\text{CLS}} =$

$$\{(x, s, y) \in \{0, 1\}^n \times \mathbb{R}_+^n \times \mathbb{R}_+^n : s_{t-1} + y_t = d_t + s_t, y_t \leq C_t x_t \text{ for } t = 1, \dots, n\}.$$

We start with the same questions.

- Q1. Is optimization over X^{CLS} polynomially solvable?
- A1. No. Capacitated lot-sizing (CLS) is shown to be \mathcal{NP} -hard in Section 6.3. Therefore, the best we can hope for is to find a good approximation of $\text{conv}(X^{\text{CLS}})$.

Q2. How do we find valid inequalities?

We again try to make use of existing inequalities, by considering relaxations for which valid inequalities are known. After looking at single periods, it is natural to aggregate together flow conservation equations from consecutive periods.

A2a. Consider the relaxation

$$\left\{ (s_{k-1}, x_k, \dots, x_l) \in \mathbb{R}_+^1 \times \{0, 1\}^{l-k+1} : s_{k-1} + \sum_{j=k}^l C_j x_j \geq \sum_{j=k}^l d_j \right\},$$

obtained by summing the constraints $s_{t-1} + y_t = d_t + s_t$ and replacing y_t by its upper bound $C_j x_t$ for periods k, \dots, l , and by replacing s_l by its lower bound of 0.

One can either attempt to generate an MIR inequality off this constraint (see Section 8.7), or else it can be viewed as a 0–1 knapsack problem with a continuous variable s_{k-1} . Using the approach used in Section 9.3, we can fix $s_{k-1} = 0$, apply a knapsack separation heuristic, and then lift back in the continuous variable.

A2b. Aggregating the same constraints as above, replacing s_k by its lower bound 0, and adding the valid inequalities (14.2) for $j = k, \dots, l$ leads to the set

$$\begin{aligned} & \{(x_k, \dots, x_l, y_k, \dots, y_l, s_l) \in \{0, 1\}^{l-k+1} \times \mathbb{R}_+^{l-k+1} \times \mathbb{R}_+^1 : \\ & \sum_{j=k}^l y_j \leq \sum_{j=k}^l d_j + s_l, y_j \leq C_j x_j, \\ & y_j \leq \left(\sum_{t=j}^l d_t \right) x_j + s_l \quad \text{for } j = k, \dots, l\}. \end{aligned}$$

Fixing $s_l = 0$, this becomes a single node flow set (see Section 9.4), for which one can generate flow cover inequalities before reintroducing s_l .

Consider the solution shown in Figure 14.4 for an instance with $n = 6$, $d = (6, 7, 4, 6, 3, 8)$, and $C = (15, 8, 10, 10, 5, 10)$.

Again because $s_3 = 0$, we can decompose the solution into two parts, one for periods 1–3 and the other for periods 4–6.

For periods 1–3, we look at the relaxation suggested in A2a, giving the set

$$\{x \in \{0, 1\}^3 : 15x_1 + 8x_2 + 10x_3 \geq 17\}$$

and the fractional point $(x_1, x_2, x_3) = \left(1, 0, \frac{1}{2}\right)$. The 0–1 knapsack separation routine, or the Chvátal–Gomory procedure for $x_1 + \frac{8}{15}x_2 + \frac{10}{15}x_3 \geq \frac{17}{15}$ both lead to the violated valid inequality

$$x_1 + x_2 + x_3 \geq 2.$$

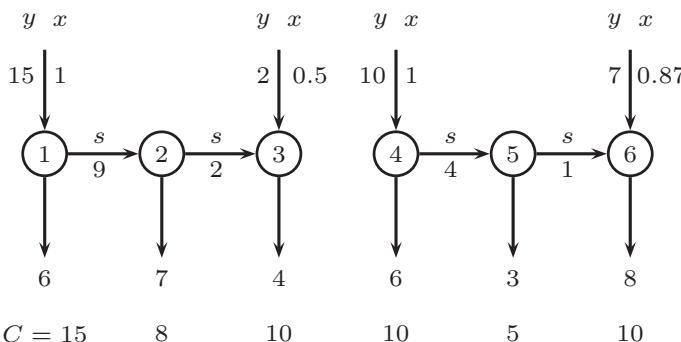


Figure 14.4 CLS solution.

For periods 4–6, we look at the relaxation suggested in A2b:

$$(x_4, x_5, x_6, y_4, y_5, y_6, s_6) \in \{0, 1\}^3 \times \mathbb{R}_+^3 \times \mathbb{R}_+^1 : y_4 + y_5 + y_6 \leq 17 + s_6,$$

$$y_4 \leq \min(10, 17)x_4 + s_6, \quad y_5 \leq \min(5, 11)x_5 + s_6, \quad y_6 \leq \min(10, 8)x_6 + s_6$$

with fractional point $(x_4, x_5, x_6, y_4, y_5, y_6, s_6) = (1, 0, 0.875, 10, 0, 7, 0)$.

The flow cover inequality with cover variables (y_4, y_6) and $\lambda = 10 + 8 - 17 = 1$ leads to the violated valid inequality

$$y_4 + y_6 + 9(1 - x_4) + 7(1 - x_6) \leq 17 + s_6.$$

Q3. Are the inequalities obtained as tight as possible?

A3. In practice, it may not be important to check that the inequalities define facets. However, it is worth checking whether the inequalities can be easily strengthened, or possibly decomposed into two or more valid inequalities.

Q4. Is the separation problem for the inequalities arising in A2a or A2b easy?

A4. In practice, a separation heuristic, such as that for flow covers, can be devised.

14.4 Multi-item Single Machine Lot-Sizing

A set of products or items have to be produced on a single machine. Because of important start-up costs and changeover times between the production of different items, the planning horizon has been broken up into equally spaced time intervals (periods such as a day, or an eight-hour shift) such that only one item is produced per period. Also certain items have minimum run-times, and others have minimum down-times specified between successive production runs of the item.

The data consists of

- the demand d_t^i for item i in period t
- the storage cost h^i per period per unit of item i
- the start-up cost g^i for item i
- the changeover cost q^{ij} when passing from the production of i to j
- the changeover time Δ^{ij} when passing from the production of i to j
- the production rate ρ^i per period for item i
- the minimum run-time α^i for item i
- the minimum down-time β^i for item i
- the production capacity C_t per period

For simplicity, we assume that the data have been preprocessed so that the initial stocks and safety stocks (lower bounds on stocks) are zero. In addition, we assume that the basic unit of each item i is the amount produced per unit of time. With this assumption, $\rho^i = 1$ for all $i = 1, \dots, m$, and $C_t = C$ is the length of a period.

We make the obvious choice of variables:

- y_t^i the amount of time that item i is produced in period t
- s_t^i the stock of i at the end of t measured in time units
- $x_t^i = 1$ if the machine is set up for i in t
- $z_t^i = 1$ if a production run of i starts in t
- $\chi_t^{ij} = 1$ if item i is produced in period $t - 1$ and j in t .

This leads to the formulation

$$\min \sum_{i=1}^m \sum_{t=1}^n \left(h^i s_t^i + g^i z_t^i + \sum_{j:j \neq i} q^{ij} \chi_t^{ij} \right) \quad (14.2)$$

$$s_{t-1}^i + y_t^i = d_t^i + s_t^i \quad \text{for all } i, t \quad (14.3)$$

$$y_t^i \leq C x_t^i - \sum_{j:j \neq i} \Delta^{ji} \chi_t^{ji} \quad \text{for all } i, t \quad (14.4)$$

$$z_t^i \geq x_t^i - x_{t-1}^i \quad \text{for all } i, t \quad (14.5)$$

$$z_t^i \leq x_t^i \quad \text{for all } i, t \quad (14.6)$$

$$z_t^i \leq 1 - x_{t-1}^i \quad \text{for all } i, t \quad (14.7)$$

$$\chi_t^{ij} \geq x_{t-1}^i + x_t^j - 1 \quad \text{for all } i, j, t \text{ with } i \neq j \quad (14.8)$$

$$\chi_t^{ij} \leq x_{t-1}^i \quad \text{for all } i, j, t \text{ with } i \neq j \quad (14.9)$$

$$\chi_t^{ij} \leq x_t^j \quad \text{for all } i, j, t \text{ with } i \neq j \quad (14.10)$$

$$z_t^i \leq x_{t+u-1}^i \quad \text{for all } i, t, u = 1, \dots, \alpha^i \quad (14.11)$$

$$z_t^i \leq 1 - x_{t-u}^i \quad \text{for all } i, t, u = 1, \dots, \beta^i \quad (14.12)$$

$$\sum_{i=1}^m x_t^i = 1 \quad \text{for all } t \quad (14.13)$$

$$s_t^i, y_t^i \geq 0, x_t^i, z_t^i \in \{0, 1\} \quad \text{for all } i, t \quad (14.14)$$

$$\chi_t^{ij} \in \{0, 1\} \quad \text{for all } i, j, t \text{ with } i \neq j \quad (14.15)$$

where (14.3) are flow conservation constraints, (14.4) are production capacity constraints, (14.5)–(14.7) define z_t^i as start-up variables, (14.8)–(14.10) define χ_t^{ij} as changeover variables, (14.11) and (14.12) express the minimum run time and minimum down-time conditions, and (14.13) the single item per period restriction. For simplicity, we ignore the fact that zero production is allowed even when an item is set up for a production run of one or several periods.

We suppose that this application is to be solved with a standard mixed integer programming system, and so no significant change of variables is envisaged. Thus, the available options are to strengthen constraints, to add valid inequalities, plus possibly knapsack and mixed integer cuts that can be generated automatically, or to use an extended formulation involving new variables.

We examine in turn various possibilities.

Improving Branch-and-Cut Performance

We present three simple observations that may radically modify the performance of a branch-and-cut algorithm.

Observation 14.2 If the variables x_t^i are all 0–1 valued and constraints (14.5)–(14.10) are present, the variables z_t^i and χ_t^{ij} will automatically take 0–1 values. Therefore, it may be possible to make the z and χ variables continuous.

Observation 14.3 If the start-up costs $\{g^i\}$ and changeover costs $\{q^{ij}\}$ are positive, consider whether it is valid and computationally interesting to drop constraints (14.6), (14.7), (14.9), and (14.10) (for the products without minimum run and down times, on the grounds that z_t^i and χ_t^{ij} will only take the value 1 if forced). Note also that (14.6) reappears as one of the constraints (14.11), and (14.7) as one of the constraints of (14.12).

Observation 14.4 Giving priorities to the binary variables x_t^i based on the period t , and/or the importance of the product i should be considered.

Problem-Specific Cutting Planes

Note first that with the relaxation $y_t^i \leq Cx_t^i$ of (14.4), the problem contains CLS relaxations for each item i , and thus the inequalities for both ULS and CLS derived in Section 14.3 can be used.

It is also natural to ask whether the basic lot-sizing inequalities can be improved in the presence of the start-up and/or changeover variables. One improvement is easily seen. The inequalities

$$s_{k-1} \geq \sum_{t=k}^l d_t(1 - x_k - \dots - x_l)$$

for ULS were valid because no production between periods k and t , which can be restated as $x_k + \dots + x_t = 0$, implies that the demand d_t must be included in the entering stock s_{k-1} . With start-up variables, the condition $x_k + z_{k+1} + \dots + z_t = 0$ also implies no production between periods k and t , and thus demonstrates the validity of

$$s_{k-1} \geq \sum_{t=k}^l d_t(1 - x_k - z_{k+1} - \dots - z_t) \quad \text{for } 1 \leq k \leq l \leq n.$$

This dominates the previous inequality as $z_i \leq x_i$ for $i = 1, \dots, n$.

Modeling Minimum Run-Times

If an item must be produced during at least α periods, there can be at most one start-up of the item in the interval $[t - \alpha + 1, t]$. In addition, if it starts at some time in this interval, then the machine must still be set up in period t . This shows that the inequality

$$\sum_{u=1}^{\alpha} z_{t-u+1} \leq y_t \quad \text{for } 1 \leq t \leq n$$

is valid for each item and can be used to replace the inequalities (14.6) and (14.11).

Similarly for the down times, the inequality

$$\sum_{u=1}^{\beta} z_{t+u} \leq 1 - y_t \quad \text{for } 1 \leq t \leq n$$

is valid, and can replace (14.7) and (14.12).

Modeling Start-ups and Changeovers

Here we consider the constraints (14.5)–(14.10) and (14.13) linking the x , z , and χ variables. First we add a small number of additional variables:

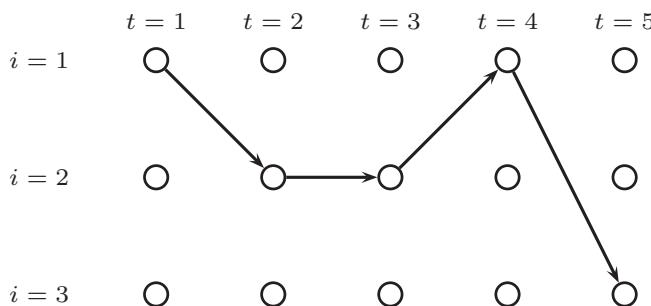


Figure 14.5 Representation of solution with changeovers.

$$\chi_t^{ij} = 1 \text{ if item } j \text{ is produced in periods } t-1 \text{ and } t.$$

Observe that if we know for which item the machine is setup in period 1, the χ variables tell us everything about the state of the machine from then on. More specifically they determine the path taken through the acyclic network shown in Figure 14.5. The path shown corresponds to the solution with $x_1^1 = x_2^2 = x_3^2 = x_4^1 = x_5^3 = 1$.

Thus we have

$$\sum_{i=1}^m x_1^i = 1$$

$$\sum_{j=1}^m \chi_t^{ij} = x_{t-1}^i \quad \text{for all } i, t$$

$$\sum_{i=1}^m \chi_t^{ij} = x_t^j \quad \text{for all } j, t$$

$$z_t^j + \chi_t^{ij} = x_t^j \quad \text{for all } j, t$$

$$x, z \in \{0, 1\}^{mn}, \chi \in \{0, 1\}^{m^2n}.$$

This formulation has an order of magnitude fewer constraints than (14.8)–(14.10), and it is as strong as possible. In what sense?

Heuristics

Relax-and-fix heuristics, in which, for example, variables in earlier periods are considered more important, provide one way of finding feasible solutions to large problems. Another approach is to consider the values of the set-up variables \tilde{x} in the linear programming solution as probabilities, and thus to construct a solution in which, for each period t , the item i produced is found by selecting i randomly with probabilities \tilde{x}_t^i for $i = 1, \dots, m$.

14.5 A Multiplexer Assignment Problem*

We consider two variants of a problem that arose in the telecommunications industry.

Problem A. Given a graph $G = (V, E)$, demands $d_e > 0$ measuring the communications required between nodes i, j where $e = (i, j) \in E$, and an unlimited number of identical rings (copies of the graph) of capacity C , the demands must be assigned unsplit to the rings without exceeding capacity. If a demand d_e with $e = (i, j)$ is assigned to a certain ring, multiplexers costing α each must be installed at nodes i and j on that ring. The problem is to find a feasible assignment of demands to rings, so that the cost/number of multiplexers installed is minimized.

Problem B. Additional constraints are imposed on Problem A based on the structure of the multiplexers. Specifically, a multiplexer contains eight slots, each of capacity $C_1 = 16$. By adding a special card to a slot, costing β , the capacity of a slot is increased to $C_2 = 63$. The problem is now to minimize the total cost of the multiplexers and the special cards.

We start by presenting a natural formulation for Problems A and B.

Let $k = 1, \dots, K$ denote the set of rings. We take as variables

$$y_e^k = 1 \text{ if demand } d_e \text{ is assigned to ring } k \text{ (NB } y_e^k \in \{0, 1\}\text{), and}$$

$$x_i^k = 1 \text{ if a multiplexer is installed at node } i \text{ of ring } k,$$

and obtain as formulation A:

$$\min \alpha \sum_{k=1}^K \sum_{i \in V} x_i^k \quad (14.16)$$

$$\sum_{k=1}^K y_e^k = 1 \quad \text{for } e \in E \quad (14.17)$$

$$\sum_{e \in E} d_e y_e^k \leq C \quad \text{for } k = 1, \dots, K \quad (14.18)$$

$$y_e^k \leq x_i^k, y_e^k \leq x_j^k \quad \text{for } e = (i, j) \in E, k = 1, \dots, K \quad (14.19)$$

$$x_i^k \in \{0, 1\} \quad \text{for } i \in V, k = 1, \dots, K \quad (14.20)$$

$$y_e^k \in \{0, 1\} \quad \text{for } e \in E, k = 1, \dots, K \quad (14.21)$$

Here constraint (14.17) assigns each demand to some ring, (14.18) ensures that the ring capacity is not exceeded, and (14.19) that multiplexers are installed where required.

In Problem B, it is necessary to add additional variables

z_i^k the number of slots of capacity C_1 used at node i on ring k , and
 w_i^k the number of slots of capacity C_2 used at node i on ring k .

The objective is now modified to become

$$\min \alpha \sum_{k=1}^K \sum_{i \in V} x_i^k + \beta \sum_{k=1}^K \sum_{i \in V} w_i^k,$$

and additional constraints

$$\begin{aligned} \sum_{e \in \delta(i)} d_e y_e^k &\leq 16z_i^k + 63w_i^k \quad \text{for } i \in V, k = 1, \dots, K \\ z_i^k + w_i^k &\leq 8x_i^k \quad \text{for } i \in V, k = 1, \dots, K \\ z_i^k, w_i^k &\in \mathbb{Z}_+^1 \quad \text{for } i \in V, k = 1, \dots, K \end{aligned}$$

are needed to ensure that enough capacity is available at each node, and that the available number of slots is not exceeded.

For these two problems, it is not clear a priori whether a branch-and-cut based MIP system has any chance of producing reasonable solutions. Thus, we first need to choose an algorithmic approach that will provide useful bounds, and then perhaps look further at the quality of the formulation. We again proceed via a series of questions. We start by examining Problem B.

Q1. Does Problem B have special structure?

A1. It is readily seen that Problem B is of the form

$$\begin{aligned} \sum_{k=1}^K y_e^k &= 1 \quad \text{for } e \in E \\ (y^k, x^k, z^k, w^k) &\in X^k \quad \text{for } k = 1, \dots, K \end{aligned}$$

with each of the sets X^k identical.

Q2. What algorithmic approach does this suggest?

A2. From Chapters 10 and 11, we might consider a Lagrangian relaxation approach or a column generation approach, where the subproblem involves optimizing over X^k , or a cutting plane approach with separation over $\text{conv}(X^k)$.

As we prefer where possible to use a general MIP system, we consider first the possibility of approximating $\text{conv}(X^k)$ with valid inequalities.

Q3. Can we find valid inequalities for $\text{conv}(X^k)$?

A3. X^k contains mixed integer knapsack constraints plus precedence constraints, but nothing else is apparent.

We also look at the complete problem.

Q4. Can we find global lower bounds on the number of multiplexers, or special slots required, either in total or at a specific node?

A4a. The total demand at node i is $D(i) = \sum_{e \in \delta(i)} d_e$. Assuming that only one multiplexer can be installed at each node on a ring, the maximum capacity of a multiplexer is $\min(C, 8 \times 63)$. Thus we obtain the valid inequality $\sum_{k=1}^K x_i^k \geq \lceil D(i)/\min(C, 504) \rceil$ for each node $i \in V$.

A4b. Working with slots in place of multiplexers, we have that

$$16 \sum_{k=1}^K z_i^k + 63 \sum_{k=1}^K w_i^k \geq D(i),$$

which after setting $Z = \sum_{k=1}^K z_i^k$ and $W = \sum_{k=1}^K w_i^k$ can be rewritten as the set $\{(Z, W) \in \mathbb{Z}_+^2 : 16Z + 63W \geq D(i)\}$. Dividing by 16, we immediately obtain the Chvátal–Gomory inequality

$$Z + 4W \geq \lceil D(i)/16 \rceil.$$

It is also simple in practice to find all the inequalities required to represent the convex hull of a set in two integer variables.

Q5. What can be done about the symmetry, namely the fact that each ring is identical, and thus many feasible solutions are essentially the same?

A5. One simple approach is to break the symmetry. As $\sum_k y_e^k = 1$ for all e , it is always possible to assign K of the edges, denoted e_1, \dots, e_K , so that e_1 is in ring 1, e_2 is in rings 1 or 2, etc. We enforce that e_κ is in one of the first κ rings by setting $y_{e_\kappa}^k = 0$ for all $k > \kappa$ and $\kappa = 1, \dots, K$.

Alternatively, the latest MIP systems will detect the symmetry between the rings and apply isomorphism pruning and/or orbital branching.

In particular, consider some node α in the branch-and-cut tree and let K_α^1 be the set of rings for which one or more of the 0–1 variables takes value 1 on the path from the root to node α . Now suppose that at node α we branch on some variable $v_p^k \in \{x_p^k, w_p^k, z_p^k\}$. We set $v_p^k = 1$ on the left branch. On the right branch, we set $v_p^k = 0$ if $k \in K_\alpha^1$, but we can set $v_p^k = 0$ for all $k \in \{1, \dots, K\} \setminus K_\alpha^1$ if $k \notin K_\alpha^1$.

Q6. What branch-and-cut options should be considered?

A6. Given the relative importance of the variables, it appears natural to give the x_i^k variables highest priority for branching followed by the w_i^k variables. The reply to Q4 also suggests the idea of explicitly introducing aggregate variables $X^i = \sum_{k=1}^K x_i^k$ with W^i and Z^i defined similarly. In this case, these variables should have higher priority than the corresponding individual variables.

Now we turn back to Problem A. As it has the same structure as Problem B, but no tightening of the formulation is apparent, decomposing as suggested in A2 appears a possibility.

Q7. Is Lagrangian relaxation or column generation a suitable algorithm for Problem A?

- A7a. Criteria for evaluating the pros and cons of using Lagrangian relaxation were given in Section 10.5. Because the rings are identical, there is only one subproblem to be solved that takes the form

$$\min \left\{ \alpha \sum_{i \in V} x_i - \sum_{e \in E} \pi_e y_e : (x, y) \in X \right\}$$

where $X = \{(x, y) \in \{0, 1\}^{|V|} \times \{0.1\}^{|E|} : \sum_{e \in E} d_e y_e \leq C, y_e \leq x_i, y_e \leq x_j \text{ for } e = (i, j) \in E\}$.

We consider the difficulty of the subproblem, the difficulty of solving the Lagrangian dual, and the strength of the dual bound. Though the subproblem is \mathcal{NP} -hard, the graphs are typically small so the subproblem should be solved quickly by branch-and-cut. The only indication of the difficulty of convergence of the subgradient algorithm is the number of dual variables $|E|$. Finally, as the constraints defining the set X certainly do not define $\text{conv}(X)$, the Lagrangian dual bound may well be significantly stronger than that provided by the linear programming relaxation of Formulation A. In practice, this turns out to be the case, and the bound is very close to the optimal integer programming value.

- A7b. Taking a column generation approach, the subproblem to be solved is the same as above, and the bound provided by the Linear Programming Master is the same as that of the Lagrangian dual. Implementing a column generation algorithm may require more work than implementing Lagrangian relaxation unless one of the codes specially designed for column generation is used. On the other hand, the column generation procedure is more robust, and less likely to have difficulties of convergence. It may also provide good feasible solutions en route.

- Q8. What about heuristics?

- A8. Problem A is an edge-partitioning problem. The simulated annealing or tabu search heuristics proposed for node partitioning in Section 13.3 can be readily adapted to this problem as feasibility of a set of edges (demands) can be tested very easily. One possibility is to take as a solution an edge partition in which the demand on each ring is feasible, and as objective function the number of multiplexers required. As neighborhood, one possibility is to take the set of feasible edge partitions obtained either by moving a single edge to another ring, or by interchanging a pair of edges lying on different rings.

14.6 Integer Programming and Machine Learning*

The possibility of using machine learning to improve IP algorithms and conversely the use of IP to tackle problems arising in data science and machine learning

are topics that have begun to attract considerable attention. Here we very briefly give some idea of the type of questions being addressed that are certainly just a beginning.

Machine Learning for IP

There have been several attempts to improve algorithms for COPs and IPs using machine learning. Aspects of algorithms that have been investigated include

- (i) branching rules that attempt to approximate strong branching, but run much faster
- (ii) rules to select a branching variable
- (iii) rules to decide whether a heuristic should be called
- (iv) rules to select which Gomory cuts should be added
- (v) a rule to decide whether a decomposition algorithm should be used on a given problem/instance.

Impressive results, with smaller branch-and-cut trees and faster solution times, have been obtained for certain problem classes: for instance if the learning is restricted to 0–1 covering problems or capacitated facility location problems. However, not surprisingly there are questions about generalizations to arbitrary MIPs, or even the effectiveness of the rules as the size of instances is significantly increased.

IP for Machine Learning

This is wide topic. Here continuous optimization has played a significant role in the training of different machine learning models. For instance linear programming has been used to train deep neural networks with various architectures and loss functions.

Other problems have been tackled by integer programming. One of the most basic questions involves binary classification. Given K data vectors of the form $(x_1^k, \dots, x_r^k, y^k)$ where $y^k \in \{0, 1\}$ gives the yes/no classification of the vector x^k , the problem is to generate a function $f(x) : x \rightarrow \{0, 1\}$ that correctly classifies new 0–1 vectors presumably drawn from the same distribution (unknown), but not in the original data set. Another approach is to generate a short set of logical clauses so as to correctly classify a set of 0–1 vectors. The exponential number of possible clauses naturally suggests a column generation algorithm. A more general classification problem is to assign a set of vectors $(x^k, y^k) \in \mathbb{R}^r \times \mathbb{R}^k$ into clusters consisting of elements with similar characteristics. One approach is to design an optimal decision tree with branches of the form $a^i x \leq b_i$ or $a^i x > b_i$ in which the leaves of the tree correspond to the clusters. In a different direction, MIP has been used to find instances that are completely misclassified by a given neural network.

14.7 Notes

- 14.1 Five of the more well-known languages are AMPL [11], GAMS [133], LINGO(LINDO) [210], MOSEL(XPRESS), and OPL(CPLEX), while the three major commercial MIP systems are CPLEX [84], GUROBI [171], and XPRESS [301]. These allow the user to develop new algorithms using callbacks. Open source systems allowing the user even greater freedom include SCIP [268] and CBC [73]. Examples of modeling languages and software systems are given in Atamtürk and Savelsbergh [18].
- 14.2 Cutting planes for ULS and CLS can be found in [250].
- 14.3 The model presented here is based on a variety of real applications. Results for models with start-ups can be found in Constantino [76]. The modeling of changeovers is from Karmarkar and Schrage [192].
- 14.4 More details on the multiplexer assignment problem and its solution appear in Sutter et al. [272] and Belvaux et al. [40].
- 14.5 The approximation of strong branching is treated in Gasse et al. [135], ways to choose branching variables have been investigated in Alvarez et al. [10] and Khalil et al. [196]. An approach to select cuts is from Tang et al. [274]. Surveys include Bengio et al. [43] and Lodi and Zarpellon [212]. Surveys on the use of optimization in machine learning include Botton et al. [58] and Gambella et al. [130]. Bienstock et al. [53] tackle the training of deep neural networks using LP. Hochbaum [178] shows how one version of the 0–1 classification problem can be solved in polynomial time as a parametric min cut problem, and Dash et al. [95] develop an IP model to provide a classifier consisting of a small set of logical clauses, while Bertsimas and Dunn's [49] use IP to design optimal decision trees. The latter authors have just published a book [50] in which IP models are used for a wide range of machine learning problems. The generation of misclassified instances for a given neural network is one of the topics in Fischetti and Jo [111].

14.8 Exercises

1. As part of a multicommodity network design problem, you encounter the subset of different commodities passing through an arc. Each of the flows has an upper bound d_j corresponding to its total demand. To meet the aggregate flow on the arc, capacity in multiples of C units needs to be installed. Thus, the set can be formulated as: $X = \{(x, y) \in \mathbb{Z} \times \mathbb{R}_+^n : \sum_{j=1}^n y_j \leq Cx, y_j \leq d_j \text{ for } j = 1, \dots, n\}$. For an instance with $n = 6$, $C = 10$, $d = (4, 9, 6, 3, 5, 7)$, you obtain the fractional solution

$y^* = (0, 9, 6, 0, 0, 7)$, $x^* = 2.2$. Find a valid inequality cutting off this point. Describe a family of valid inequalities for the set X .

2. Consider a unit commitment problem with 5 generators and 12 (two-hour) time periods. Period 1 follows on again from period 12, and the pattern is repeated daily. $d = (50, 60, 50, 100, 80, 70, 90, 60, 50, 120, 110, 70)$ are the demands per period, and the reserve is 1.2 times the demand, so the total capacity of the generators switched on in any period must be at least this reserve. The capacity of the generators is $C = (12, 12, 35, 50, 75)$, and their minimum levels of production are $L = (2, 2, 5, 20, 40)$. In addition each generator must stay on for at least two periods. The ramping constraints only apply to the fifth generator – when on in two successive periods, the output cannot increase by more than 20 from one period to the next, and cannot decrease by more than 15. The costs are approximate. The main cost is a start-up cost $g = (100, 100, 300, 400, 800)$. There are also fixed costs $f = (1, 1, 5, 10, 15)$ and variables costs $p = (10, 10, 4, 3, 2)$ in each period that a generator is on.

Formulate and solve with a mixed integer programming system.

3. Consider an item as in the multi-item lot-sizing problem in Section 14.4.
 - (i) Suppose that the item cannot be produced for more than γ consecutive periods. Formulate. If the formulation appears to be weak, look for a stronger formulation.
 - (ii) If the item must be produced at least once every δ periods, give a strong formulation.
4. Suppose that $x^1, x^2 \in \{0, 1\}^n$ and x^1 is lexicographically greater than or equal to x^2 . Formulate and demonstrate the validity of your formulation.
5. You have a discrete time model in which $x_t^i = 1$ if job i starts in period t . All three jobs must be carried out. Given two jobs, you wish to formulate the constraint that the start of job 1 is not more than three periods after the start of job 2 and the start of job 3 is at least five periods after the start of job 1.
 - (i) Formulate without introducing additional variables.
 - (ii) Let $z_t^i = 1$ if job i starts in or before period t . Reformulate using the z_t^i variables. Can you say anything about the structure of the constraint matrix?
6. Find a minimum cost directed Steiner tree in the complete digraph on $n = 10$ nodes with root node $r = 1$, terminal nodes $2, 5, 7, 8, 10$, and arc costs

$$\begin{pmatrix} 12 & 6 & 7 & 4 & 11 & 8 & 4 & 5 & 7 & 15 \\ 9 & 4 & 2 & 26 & 12 & 4 & 7 & 11 & 4 & 28 \\ 6 & 3 & 8 & 2 & 12 & 15 & 19 & 3 & 7 & 9 \\ 21 & 33 & 24 & 52 & 2 & 19 & 6 & 2 & 9 & 15 \\ 6 & 3 & 5 & 8 & 4 & 3 & 2 & 7 & 4 & 12 \\ 14 & 17 & 32 & 24 & 15 & 11 & 22 & 28 & 9 & 6 \\ 8 & 3 & 5 & 2 & 3 & 4 & 7 & 8 & 10 & 3 \\ 31 & 24 & 46 & 52 & 43 & 13 & 24 & 27 & 61 & 21 \\ 2 & 3 & 4 & 3 & 5 & 7 & 9 & 3 & 5 & 9 \\ 21 & 24 & 13 & 38 & 67 & 94 & 24 & 3 & 26 & 23 \end{pmatrix}.$$

7. Consider an instance of a constant capacity lot-sizing model with $n = 4$ periods, an initial stock of eight units, demands $d = (2, 3, 6, 5)$, lower bounds on stocks of $(5, 2, 3, 0)$ units, and production capacity $C = 5$. Convert the instance to an equivalent one with no initial stock and no zero lower bounds on the stocks. Describe an algorithm for the general case.

8. * Show that the inequality

$$\sum_{j \in S} y_j \leq \sum_{j \in S} \left(\sum_{i=j}^l d_i \right) x_j + s_l$$

is facet-defining for X^{ULS} where $1 \leq l \leq n$ and $S \subseteq \{1, \dots, l\}$. When is the more general inequality of Exercise 7 in Chapter 8 facet-defining?

9. Use the fact that any $s-t$ flow can be represented as a sum of flows on individual $s-t$ paths to reformulate the fixed charge network flow problem of Exercise 10 in Chapter 3. Does this lead to any new algorithms for the problem?
10. A Multiplexer Assignment Instance. Five nodes (communications centers) are linked on a network. Demands between nodes are

$$(d_e) = \begin{pmatrix} - & 86 & 45 & 65 & 116 \\ - & - & 9 & 0 & 41 \\ - & - & - & 104 & 126 \\ - & - & - & - & 8 \\ - & - & - & - & - \end{pmatrix}.$$

An unlimited number of fiber optic cables can be run through the five nodes in a ring. If part of the demand between nodes i and j is placed on some ring, add/drop multiplexers (ADMs) must be placed at nodes i and j . The total

demand assigned to a ring (assuming 100% protection) cannot exceed 256. Each ADM has eight slots. The capacity of each slot is 16, unless a terminal multiplexer (TM) is installed in the slot, in which case its capacity is 63. The cost of an ADM is 3 and of a TM is 1. Find an assignment of the demands to the rings that minimizes the total cost.

- 11.** The Traveling Salesman Problem with Time Windows. A truck driver must deliver to nine customers on a given day, starting and finishing at the depot. Each customer $i = 1, \dots, 9$ has a time window $[r_i, d_i]$ and an unloading time p_i . The driver must start unloading at client i during the specified time interval. If she is early, she has to wait till time r_i before starting to unload. Node 0 denotes the depot, and c_{ij} the time to travel between nodes i and j for $i, j \in \{0, 1, \dots, 9\}$. The data are $p = (0, 1, 5, 9, 2, 7, 5, 1, 5, 3)$, $r = (0, 2, 9, 4, 12, 0, 23, 9, 15, 10)$, $d = (150, 45, 42, 40, 150, 48, 96, 100, 127, 66)$, and

$$(c_{ij}) = \begin{pmatrix} - & 5 & 4 & 4 & 4 & 6 & 3 & 2 & 1 & 8 \\ 7 & - & 2 & 5 & 3 & 5 & 4 & 4 & 4 & 9 \\ 3 & 4 & - & 1 & 1 & 12 & 4 & 3 & 11 & 6 \\ 2 & 2 & 3 & - & 2 & 23 & 2 & 9 & 11 & 4 \\ 6 & 4 & 7 & 2 & - & 9 & 8 & 3 & 2 & 1 \\ 1 & 4 & 6 & 7 & 3 & - & 8 & 5 & 7 & 4 \\ 12 & 32 & 5 & 12 & 18 & 5 & - & 7 & 9 & 6 \\ 9 & 11 & 4 & 12 & 32 & 5 & 12 & - & 5 & 22 \\ 6 & 4 & 7 & 3 & 5 & 8 & 6 & 9 & - & 5 \\ 4 & 6 & 4 & 7 & 3 & 5 & 8 & 6 & 9 & - \end{pmatrix}.$$

- 12.** Lot-Sizing with Minimum Batch Sizes and Cleaning Times. Consider the problem of production of a single item. Demands $\{d_t\}_{t=1}^n$ are known in advance. Production capacity in each time period is C , but in the last period of a production sequence it is reduced to \tilde{C} . In each production sequence, at least a minimum amount P must be produced, and production is at full capacity in all but the first and last periods of a production sequence. There is a fixed cost of f for each period in which the item is produced, the storage costs are h per item per period, and the backlogging costs g . Solve an instance with $n = 12$, $d = (5, 4, 0, 0, 6, 3, 2, 0, 0, 4, 9, 0)$, $C = 7$, $\tilde{C} = 4$, $f = 50$, $h = 1$, $g = 10$, and $P = 19$.

References

- 1** Aarts, E.H.L. and Lenstra, J.K. (eds.) (1997). Editors of the book *Local Search in Combinatorial Optimization*. Chichester: Wiley.
- 2** Absi, N., Archetti, C., Dauzère-Pérès, S., and Feillet, D. (2015). A two-phase iterative heuristic approach for the production routing problem. *Transportation Science* 49 (4): 784–795.
- 3** Achterberg, T. (2009). SCIP: solving constraint integer programs. *Mathematical Programming Computation* 1 (1): 1–41.
- 4** Achterberg, T. and Berthold, T. (2007). Improving the feasibility pump. *Discrete Optimization* 4: 77–86.
- 5** Achterberg, T., Bixby, R.E., Gu, Z. et al. (2019). Presolve reductions in mixed integer programming. *INFORMS Journal on Computing* 32 (2) 1–34.
- 6** Achterberg, T., Koch, T., and Martin, A. (2004). Branching rules revisited. *Operations Research Letters* 33: 42–54.
- 7** Achterberg, T. and Wunderling, R. (2013). Mixed integer programming: analyzing 12 years of progress. In: *Facets of Combinatorial Optimization* (ed. M. Junger and G. Reinelt), 449–481. Berlin: Springer-Verlag.
- 8** Aghezzaf, E.H., Magnanti, T.L., and Wolsey, L.A. (1995). Optimizing constrained subtrees of trees. *Mathematical Programming* 71: 113–126.
- 9** Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. (1993). *Network Flows*. Englewood Cliffs, NJ: Prentice Hall.
- 10** Alvarez, A.M., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal on Computing* 29 (1): 185–195.
- 11** AMPL (2003). AMPL modeling language. <https://ampl.com> (accessed 20 April 2020).
- 12** Applegate, D.L., Bixby, R.E., Chvátal, V., and Cook, W.J. (2007). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.
Companion website: www.wiley.com/go/wolsey/integerprogramming2e

- 13** Applegate, D., Cook, W.J., Bixby, R., and Chvátal, V. (1995). Finding Cuts in the TSP. *DIMACS Technical Report 95-05*. New Brunswick, NJ: Rutgers University.
- 14** Atamtürk, A. (2001). Flow pack facets for the single node fixed charge flow polytope. *Operations Research Letters* 29: 107–114.
- 15** Atamtürk, A. and Günlük, O. (2010). Mingling: mixed integer rounding with bounds. *Mathematical Programming* 123: 315–338.
- 16** Atamtürk, A., Küçükyavuz, S., and Tezel, B. (2018). Path cover and path pack inequalities for the capacitated fixed charge network flow problem. *SIAM Journal on Optimization* 27 (3): 1943–1976.
- 17** Atamtürk, A., Nemhauser, G.L., and Savelsbergh, M.W.P. (2000). Conflict graphs in solving integer programming problems. *European Journal of Operational Research* 121 (1): 40–55.
- 18** Atamtürk, A. and Savelsbergh, M.W.P. (2005). Integer programming software systems. *Annals of Operations Research* 140: 67–124.
- 19** Balakrishnan, A., Magnanti, T.L., and Wong, R.T. (1995). A decomposition algorithm for local access telecommunication network expansion planning. *Operations Research* 43: 58–66.
- 20** Balas, E. (1965). An additive algorithm for solving linear programs with 0-1 variables. *Operations Research* 13: 517–546.
- 21** Balas, E. (1975). Disjunctive programs: cutting planes from logical conditions. In: *Nonlinear Programming*, vol. 2 (ed. O.L. Mangasarian R.R. Meyer and S.M. Robinson), 279–312. New York: Academic Press.
- 22** Balas, E. (1975). Facets of the knapsack polytope. *Mathematical Programming* 8: 146–164.
- 23** Balas, E. (1989). The prize collecting traveling salesman problem. *Networks* 19: 621–636.
- 24** Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research* 86: 529–558.
- 25** Balas, E., Ceria, S., and Cornuéjols, G. (1993). A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming* 58: 295–324.
- 26** Balas, E., Ceria, S., and Cornuéjols, G. (1996). Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science* 42: 1229–1246.
- 27** Balas, E., Ceria, S., Cornuéjols, G., and Natraj, G. (1996). Gomory cuts revisited. *Operations Research Letters* 19: 1–9.
- 28** Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming* 115 (2): 351–385.

- 29** Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research* 59 (5): 1269–1283.
- 30** Ball, M.O., Magnanti, T.L., Monma, C.L., and Nemhauser, G.L. (eds.) (1995). *Network Models, Handbooks in Operations Research and Management Science*, vol. 7. Amsterdam: North-Holland.
- 31** Ball, M.O., Magnanti, T.L., Monma, C.L., and Nemhauser, G.L. (eds.) (1995). *Network Routing, Handbooks in Operations Research and Management Science*, vol. 8. Amsterdam: North-Holland.
- 32** Barahona, F. and Anbil, R. (2000). The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming* 87: 385–399.
- 33** Barány, I., Edmonds, J., and Wolsey, L.A. (1986). Packing and covering a tree by subtrees. *Combinatorica* 6: 245–257.
- 34** Barnhart, C., Johnson, E.L., Nemhauser, G.L., and Savelsbergh, M.W.P. (1998). Branch and price: column generation for solving huge integer programs. *Operations Research* 46 (3): 316–329.
- 35** Bauer, P. (1997). The circuit polytope: facets. *Mathematics of Operations Research* 22: 110–145.
- 36** Beale, E.M.L. (1979). Branch and bound methods for mathematical programming systems. *Annals of Discrete Mathematics* 5: 201–219.
- 37** Beale, E.M.L. and Tomlin, J.A. (1970). Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. In: *Proceedings of the 5th Annual Conference on Operational Research* (ed. J. Lawrence), 447–454. Tavistock Publications.
- 38** Beasley, J.E. (1993). Lagrangean relaxation. In: *Modern Heuristic Techniques for Combinatorial Problems*, Chapter 6 (ed. C.R. Reeves), 243–303. Oxford: Blackwell Scientific Publications.
- 39** Bellman, R.E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.
- 40** Belvaux, G., Boissin, N., Sutter, A., and Wolsey, L.A. (1998). Optimal placement of add/drop multiplexers: static and dynamic models. *European Journal of Operational Research* 108: 26–35.
- 41** Benders, J.F. (1962). Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik* 4: 238–252.
- 42** Ben-Ameur, W. and Neto, J. (2007). Acceleration of cutting-plane and column generation algorithms: applications to network design. *Networks* 49 (1): 3–17.
- 43** Bengio, Y., Lodi, A., and Prouvost, A. (2018). Machine learning for combinatorial optimization: a methodological tour d’horizon. arXiv:1811.06128v1 [cs.LG], 15 November 2018.

- 44** Berge, C. (1957). Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America* 43: 842–844.
- 45** Berge, C. (1973). *Graphs and Hypergraphs*. Amsterdam: North-Holland.
- 46** Berthold, T. and Hendel, G. (2015). Shift-and-propagate. *Journal of Heuristics* 21: 73–106.
- 47** Bertold, T., Perregard, M., and Meszaros, C. (2018). Four good reasons to use an interior point solver within a MIP solver. In: *Operations Research Proceedings 2017* (ed. N. Kliewer, J.F. Ehmke, and R. Borndorfer), 159–164. Springer.
- 48** Bertsekas, D. (2017). *Dynamic Programming and Optimal Control*, vol. I, 4e. Athena Scientific.
- 49** Bertsimas, D. and Dunn, J. (2017). Optimal classification trees. *Machine Learning* 106: 1039–1082.
- 50** Bertsimas, D. and Dunn, J. (2019). *Machine Learning Under a Modern Optimization Lens*. Dynamic Ideas LLC.
- 51** Bertsimas, D. and Weismantel, R. (2005). *Optimization Over Integers*. Belmont, MA: Dynamic Ideas LLC.
- 52** Bienstock, D. and Günlük, O. (1996). Capacitated network design: polyhedral structure and computation. *ORSA Journal on Computing* 8 (3): 243–259.
- 53** Bienstock, D., Munoz, G., and Pokutta, S. (2018). Principled deep neural network training through linear programming. arXiv:1810.0321 v2 [cs.LG].
- 54** Bixby, R.E., Fenelon, M., Gu, Z. et al. (2004). Mixed-integer programming: a progress report. In: *The Sharpest Cut: The Impact of Manfred Padberg and His Work*, Chapter 18 (ed. M. Grotschel), 309–325. Philadelphia, PA: SIAM.
- 55** Bodur, M., Dash, S., Günlük, O., and Luedtke, J. (2017). Strengthened Benders cuts for stochastic integer programs with continuous recourse. *INFORMS Journal on Computing* 29: 77–91.
- 56** Bonami, P. (2012). On optimizing over lift-and-project closures. *Mathematical Programming Computation* 4: 151–179.
- 57** Bondy, J.A. and Murty, U.S.R. (1976). *Graph Theory with Applications*. London: Macmillan.
- 58** Bottou, L., Curtis, F.E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review* 60 (2): 223–311.
- 59** Brahimi, N. and Dauzère-Pérès, S. (2015). A Lagrangian heuristic for capacitated single item lot-sizing problems. *4OR* 13 (2): 173–198.
- 60** Brandao, F. and Pedroso, J.P. (2016). Bin packing and related problems: general arc-flow formulation with graph compression. *Computers & Operations Research* 69: 56–67.
- 61** Brearley, A.L., Mitra, G., and Williams, H.P. (1973). An analysis of mathematical programs prior to applying the simplex method. *Mathematical Programming* 7: 263–282.

- 62** Briant, O., Lemaréchal, C., Meurdesoif, P. et al. (2008). Comparison of bundle and classical column generation. *Mathematical Programming* 113: 299–344.
- 63** Caprara, A. and Fischetti, M. (1997). Branch and cut algorithms. In: *Annotated Bibliographies in Combinatorial Optimization*, Chapter 4 (ed. M. Dell'Amico, F. Maffioli, and S. Martello), 45–64. Chichester: Wiley.
- 64** Caroe, C.C. and Tind, J. (1998). L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming* 83: 451–464.
- 65** Ceria, S., Cordier, C., Marchand, H., and Wolsey, L.A. (1998). Cutting planes for integer programs with general integer variables. *Mathematical Programming* 81: 201–214.
- 66** Černý, V. (1985). A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45: 41–51.
- 67** Christofides, N. (1976). Worst Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Report 388*. GSIA, Carnegie-Mellon University.
- 68** Christofides, N., Mingozzi, A., and Toth, P. (1981). State space relaxation procedures for the computation of bounds to routing problems. *Networks* 11: 145–164.
- 69** Christofides, N., Mingozzi, A., and Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming* 20: 255–282.
- 70** Chvátal, V. (1973). Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics* 4: 305–337.
- 71** Chvátal, V. (1983). *Linear Programming*. New York: Freeman.
- 72** Clochard, J.M. and Naddef, D. (1993). Using path inequalities in a branch and cut code for the symmetric travelling salesman problem. In: *Proceedings 3rd IPCO Conference* (ed. G. Rinaldi and L.A., Wolsey), 291–311. Louvain-la-Neuve, CIACO.
- 73** COIN-OR branch-and-cut, or CBC User Guide. <https://www.coin-or.org/Cbc/> (accessed 21 April 2020).
- 74** Conforti, M., Cornuéjols, G., and Zambelli, G. (2014). *Integer Programming*. Heidelberg: Springer-Verlag.
- 75** Conforti, M. and Wolsey, L.A. (2019). “Facet” separation with one linear program. *Mathematical Programming* 178 (1): 361–380.
- 76** Constantino, M. (1996). A cutting plane approach to capacitated lot-sizing with start-up costs. *Mathematical Programming* 75: 353–376.
- 77** Constantino, M. (1998). Lower bounds in lot-sizing models: a polyhedral study. *Mathematics of Operations Research* 23: 101–118.
- 78** Cook, S.A. (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing Machinery*, Shaker Heights, OH: ACM, pp. 151–158.

- 79** Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., and Schrijver, A. (1997). *Combinatorial Optimization*. New York: Wiley.
- 80** Cook, W.J., Kannan, R., and Schrijver, A. (1990). Chvátal closures for mixed integer programming problems. *Mathematical Programming* 47: 155–174.
- 81** Cook, W., Rutherford, T., Scarf, H.E., and Shallcross, D. (1993). An implementation of the generalized basis reduction algorithm for integer programming. *ORSA Journal on Computing* 5: 206–212.
- 82** Cornuéjols, G., Sridharan, R., and Thizy, J.M. (1991). A comparison of heuristics and relaxations for the capacitated plant location problem. *European Journal of Operational Research* 50: 280–297.
- 83** Costa, L., Contardo, C., and Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science* 53 (4): 946–985.
- 84** Cplex (2020). IBM ILOG CPLEX optimization studio. <https://www.ibm.com/ibm/products/ilog-cplex-optimization-studio> (accessed 21 April 2020).
- 85** Crescenzi, P. and Kann, V. (2005) A compendium of \mathcal{NP} optimization problems. <http://www.csc.kth.se/viggo/wwwcompendium/> (accessed 21 April 2020).
- 86** Crowder, H., Johnson, E.L., and Padberg, M.W. (1983). Solving large scale zero-one linear programming problems. *Operations Research* 31: 803–834.
- 87** Cunha, J.S., Simonetti, L., and Lucena, A. (2016). Lagrangian heuristics for the quadratic knapsack problem. *Computational Optimization and Applications* 63: 97–120.
- 88** Dakin, R.J. (1965). A tree search algorithm for mixed integer programming problems. *Computer Journal* 8: 250–255.
- 89** Danna, E., Rothberg, E., and Pape, C.L. (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming* 102: 71–90.
- 90** Dantzig, G.B. (1957). Discrete variable extremum problems. *Operations Research* 5: 266–277.
- 91** Dantzig, G.B. (1963). *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press.
- 92** Dantzig, G.B., Ford, L.R., and Fulkerson, D.R. (1956). A primal-dual algorithm for linear programming. In: *Linear Inequalities and Related Systems* (ed. H. Kuhn and A.W. Tucker), 171–182. Princeton, NJ: Princeton University Press.
- 93** Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M. (1954). Solution of a large scale traveling salesman problem. *Operations Research* 2: 393–410.
- 94** Dantzig, G.B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research* 8: 101–111.

- 95** Dash, S., Günlük, O., and Wei, D. (2018). Boolean decision rules via column generation. arXiv:1805.09901v1[cs:AI], May 2018.
- 96** Desaulniers, G., Desrosiers, J., and Solomon, M.M. (2005). *Column Generation*. Springer.
- 97** Desrosiers, J., Dumas, Y., Solomon, M.M., and Soumis, F. (1995). Time constrained routing and scheduling. In: *Network Routing, Handbooks in Operations Research and Management Science*, Chapter 2, vol. 8 (ed. M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser), 35–139. Amsterdam: North-Holland.
- 98** Desrosiers, J. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science* 23: 1–13.
- 99** Desrosiers, J., Soumis, F., and Desrochers, M. (1984). Routing with time windows by column generation. *Networks* 14: 545–565.
- 100** Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. Cambridge, MA: MIT Press.
- 101** Edmonds, J. (1965). Paths, trees and flowers. *Canadian Journal of Mathematics* 17: 449–467.
- 102** Edmonds, J. (1965). Maximum matching and a polyhedron with 0-1 vertices. *Journal of Research of the National Bureau of Standards* 69B: 125–130.
- 103** Edmonds, J. (1970). Submodular functions, matroids and certain polyhedra. In: *Combinatorial Structures and Their Applications, Proceedings of the Calgary International Conference* (ed. R. Guy), 69–87. Gordon and Breach.
- 104** Edmonds, J. (1971). Matroids and the greedy algorithm. *Mathematical Programming* 1: 127–136.
- 105** Edmonds, J. and Giles, R. (1997). A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics* 1: 185–204.
- 106** Eksioglu, S.D., Eksioglu, B., and Romijn, H.E. (2007). A Lagrangean heuristic for integrated production and transportation planning problems in a dynamic, multi-item, two-layer supply chain. *IIE Transactions* 39 (2): 191–201.
- 107** Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location. *Operations Research* 26: 992–1009.
- 108** Everett, H. III (1963). Generalized Lagrange multiplier method for solving problems of optimal allocation of resources. *Operations Research* 11: 399–417.
- 109** Feo, T.A. and Resende, M.G.C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6: 109–134.
- 110** Fiorini, S., Massar, S., Pokutta, S. et al. (2015). Exponential lower bounds for polytopes in combinatorial optimization. *Journal of the ACM* 62 (2): 1–23.
- 111** Fischetti, M. and Jo, J. (2018). Deep neural networks and mixed integer linear optimization. *Constraints* 23: 296–309.
- 112** Fischetti, M., Glover, F., and Lodi, A. (2005). The feasibility pump. *Mathematical Programming* 104: 91–104.

- 113** Fischetti, M., Ljubic, I., and Sinnl, M. (2016). Benders' decomposition without separability: a computational study for capacitated facility location problems. *European Journal of Operational Research* 253: 557–569.
- 114** Fischetti, M., Ljubic, I., and Sinnl, M. (2017). Redesigning Benders' decomposition for large scale facility location. *Management Science* 63 (7): 2146–2162.
- 115** Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical Programming* 98: 23–47.
- 116** Fischetti, M. and Salvagnin, D. (2009). Feasibility pump 2.0. *Mathematical Programming Computation* 1: 201–222.
- 117** Fischetti, M. and Salvagnin, D. (2010). An in-out approach to disjunctive optimization. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science*, vol. 6140, Proceedings of the 7th International Conference, CPAIOR 2010, Bologna, Italy (14–18 June 2010) (ed. A. Lodi, M. Milano, and P. Toth), 136–140. Springer.
- 118** Fischetti, M., Salvagnin, D., and Zanette, A. (2010). A note on the selection of Benders' cuts. *Mathematical Programming* 124: 175–182.
- 119** Fisher, M.L. (1981). The Lagrangean relaxation method for solving integer programming problems. *Management Science* 27: 1–18.
- 120** Fisher, M.L. and Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks* 11 (2): 109–124.
- 121** Fisher, M.L. and Rinnooy Kan, A.H.G. (eds.) (1988). The design, analysis and implementation of heuristics. *Management Science* 34: 263–429.
- 122** Fleischmann, D. (1990). The discrete lot-sizing and scheduling problem. *European Journal of Operational Research* 44: 337–348.
- 123** Florian, M. and Klein, M. (1971). Deterministic production planning with concave costs and capacity constraints. *Management Science* 18: 12–20.
- 124** Ford, L.R. Jr. and Fulkerson, D.R. (1956). Maximum flow through a network. *Canadian Journal of Mathematics* 8: 399–404.
- 125** Ford, L.R. Jr. and Fulkerson, D.R. (1962). *Flows in Networks*. Princeton, NJ: Princeton University Press.
- 126** Frangioni, A., Gendron, B., and Gorgone, E. (2017). On the computational efficiency of subgradient methods: a case study with Lagrangian bounds. *Mathematical Programming Computation* 9 (4): 573–604.
- 127** Fukasawa, R., Longo, H., Lysgaard, J. et al. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming* 106: 491–511.
- 128** Fulkerson, D.R. and Gross, D.A. (1965). Incidence matrices and interval graphs. *Pacific Journal of Mathematics* 15: 833–835.

- 129** Gade, D., Küçükyavuz, S., and Sen, S. (2014). Decomposition algorithms with parametric Gomory cuts for two-stage stochastic integer programs. *Mathematical Programming* 144: 39–64.
- 130** Gambella, C., Ghaddar, B., and Naoum-Sawaya, J. (2019). Optimization problems for machine learning: a survey. arXiv:1901.05331v3 [math OC], December 2019.
- 131** Gamrath, G. (2010). Generic branch-cut-and-price, Diplomarbeit, 1 Fachbereich Mathematik der Technischen Universität Berlin.
- 132** Gamrath, G., Koch, T., Martin, A. et al. (2015). Progress in presolving for mixed integer programming. *Mathematical Programming Computation* 7 (4): 367–398.
- 133** GAMS (2020). Modeling language. <https://gams.com> (accessed 21 April 2020).
- 134** Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman.
- 135** Gasse, M., Chételat, D., Ferroni, N. et al. (2019). Exact combinatorial optimization with graph convolutional neural networks. arXiv:1906.01629v2 [cs.LG], June 2019.
- 136** Geoffrion, A.M. (1972). Generalized Benders decomposition. *Journal of Optimization Theory and Applications* 10: 237–260.
- 137** Geoffrion, A.M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study* 2: 82–114.
- 138** Geoffrion, A.M. and Graves, G.W. (1974). Multicommodity distribution design by Benders decomposition. *Management Science* 20: 822–844.
- 139** Geoffrion, A.M. and Marsten, R.E. (1972). Integer programming algorithms: a framework and state-of-the art survey. *Management Science* 18: 465–491.
- 140** Ghouila-Houri, A. (1962). Caractérisation des matrices totalement unimodulaires. *Comptes rendus de l'Académie des Sciences* 254: 1192–1194.
- 141** Gilmore, P.C. and Gomory, R.E. (1961). A linear programming approach to the cutting stock problem. *Operations Research* 9: 849–859.
- 142** Gilmore, P.C. and Gomory, R.E. (1963). A linear programming approach to the cutting stock problem: Part II. *Operations Research* 11: 863–888.
- 143** Gilmore, P.C. and Gomory, R.E. (1966). The theory and computation of knapsack functions. *Operations Research* 14: 1045–1074.
- 144** Gleixner, A., Bastubbe, M., Eifler, L. et al. (2018). The SCIP Optimization Suite 6.0. *ZIB-report* 18-26. Zuse Institute Berlin.
- 145** Glover, F. (1968). Surrogate constraints. *Operations Research* 16: 741–749.
- 146** Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research* 13: 533–549.
- 147** Glover, F. (1989). Tabu search: Part I. *ORSA Journal on Computing* 1: 190–206.
- 148** Glover, F. (1990). Tabu search: Part II. *ORSA Journal on Computing* 2: 4–32.

- 149 Goemans, M.X. (1994). The Steiner polytope and related polyhedra. *Mathematical Programming* 63: 157–182.
- 150 Goffin, J.-L. (1977). On the convergence rates of subgradient optimization methods. *Mathematical Programming* 13: 329–347.
- 151 Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.
- 152 Golden, B., Raghavan, S., and Wasil, E. (eds.) (2008). *The Vehicle Routing Problem*. Springer.
- 153 Gomory, R.E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* 64: 275–278.
- 154 Gomory, R.E. (1960). An Algorithm for the Mixed Integer Problem. RM-2597. The Rand Corporation.
- 155 Gomory, R.E. (1963). An algorithm for integer solutions to linear programs. In: *Recent Advances in Mathematical Programming* (ed. R. Graves and P. Wolfe), 269–302. New York: McGraw-Hill.
- 156 Gomory, R.E. (1965). On the relation between integer and non-integer solutions to linear programs. *Proceedings of the National Academy of Sciences of the United States of America* 53: 260–265.
- 157 Gomory, R.E. (1969). Some polyhedra related to corner problems. *Linear Algebra and its Applications* 2: 451–588.
- 158 Gomory, R.E. and Hu, T.C. (1961). Multi-terminal network flows. *SIAM Journal* 9: 551–570.
- 159 Gomory, R.E. and Johnson, E.L. (1972). Some continuous functions related to corner polyhedra. *Mathematical Programming* 3: 23–85.
- 160 Graham, R.L. (1966). Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal* 45: 1563–1581.
- 161 Gröflin, H. and Liebling, T. (1981). Connected and alternating vectors: polyhedra and algorithms. *Mathematical Programming* 20: 233–244.
- 162 Grötschel, M., Lovász, L., and Schrijver, A. (1981). The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1: 169–197.
- 163 Grötschel, M., Lovász, L., and Schrijver, A. (1984). Corrigendum to our paper “The ellipsoid method and its consequences in combinatorial optimization”. *Combinatorica* 4: 291–295.
- 164 Grötschel, M., Lovasz, L., and Schrijver, A. (1988). *Geometric Algorithms and Combinatorial Optimization*. Berlin: Springer-Verlag.
- 165 Grötschel, M. and Padberg, M.W. (1975). Partial linear characterizations of the asymmetric travelling salesman polytope. *Mathematical Programming* 8: 378–381.
- 166 Grunbaum, B. (1967). *Convex Polytopes*. New York: Wiley.

- 167 Gu, Z., Nemhauser, G.L., and Savelsbergh, M.W.P. (1999). Lifted flow cover inequalities for mixed 0–1 integer programs. *Mathematical Programming* 85: 439–467.
- 168 Guignard, M. and Kim, S. (1987). Lagrangean decomposition for integer programming: theory and applications. *RAIRO - Operations Research* 21: 307–323.
- 169 Guignard, M. and Spielberg, K. (1981). Logical reduction methods in zero-one programming. *Operations Research* 29: 49–74.
- 170 Günlük, O. and Pochet, Y. (2001). Mixing mixed-integer inequalities. *Mathematical Programming* 90 (3): 429–457.
- 171 GUROBI. <https://www.gurobi.com> (accessed 21 April 2020).
- 172 Hammer, P.L., Johnson, E.L., and Peled, U.N. (1975). Facets of regular 0-1 polytopes. *Mathematical Programming* 8: 179–206.
- 173 Held, M. and Karp, R.M. (1962). A dynamic programming approach to sequencing problems. *Journal of SIAM* 10: 196–210.
- 174 Held, M. and Karp, R.M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research* 18: 1138–1162.
- 175 Held, M. and Karp, R.M. (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming* 1: 6–25.
- 176 Held, M., Wolfe, P., and Crowder, H.P. (1974). Validation of subgradient optimization. *Mathematical Programming* 6: 62–88.
- 177 Hendel, G. (2018). Adaptive Large Neighbourhood Search for Mixed Integer Programs. *ZIB Report* 18-60. Berlin: Konrad-Zuse Institute.
- 178 Hochbaum, D.S. (2018). *Machine Learning and Data Mining with Combinatorial Optimization Algorithms*, 109–129. INFORMS TutORials in Operations Research. Published online: 19 October 2018.
- 179 Hoffman, A.J. and Kruskal, J.B. (1956). Integral boundary points of convex polyhedra. In: *Linear Inequalities and Related Systems* (ed. H.W. Kuhn and A.W. Tucker), 223–246. Princeton, NJ: Princeton University Press.
- 180 Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.
- 181 Ibarra, O.H. and Kim, C.E. (1975). Fast approximations algorithms for the knapsack and sum of subset problems. *Journal of the ACM* 22: 463–468.
- 182 Jena, S.D., Cordeau, J.-F., and Gendron, B. (2017). Lagrangean heuristics for large-scale dynamic facility location problems with generalized modular capacities. *Informs Journal of Computing* 29 (3): 388–404.
- 183 Jepsen, M., Petersen, B., Spoerrendonk, S., and Pisinger, D. (2008). Subset row inequalities applied to the vehicle routing problem with time windows. *Operations Research* 56 (2): 497–511.

- 184** Jepsen, M., Petersen, B., Spoerrendonk, S., and Pisinger, D. (2014). A branch-and-cut algorithm for the capacitated profitable tour problem. *Discrete Optimization* 14: 78–96.
- 185** Jeroslow, R.G. (1972). Cutting plane theory: disjunctive methods. *Annals of Discrete Mathematics* 1: 293–330.
- 186** Johnson, E.L. (1980). *Integer Programming - Facets, Subadditivity and Duality for Group and Semigroup Problems*. Philadelphia, PA: SIAM Publications.
- 187** Johnson, D.S., Demers, A., Ullman, J.D. et al. (1974). Worst case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing* 3: 299–325.
- 188** Johnson, E.L., Mehrotra, A., and Nemhauser, G.L. (1993). Min-cut clustering. *Mathematical Programming* 62: 133–152.
- 189** Jornsten, K. and Nasberg, M. (1986). A new Lagrangian relaxation approach to the generalized assignment problem. *European Journal of Operational Research* 27: 313–323.
- 190** Junger, M., Liebling, T., Naddef, D. et al. (eds.) (2010). *50 Years of Integer Programming 1958–2008*. Heidelberg: Springer-Verlag.
- 191** Jünger, M., Reinelt, G., and Thienel, S. (1995). Practical problem solving with cutting plane algorithms in combinatorial optimization. In: *Combinatorial Optimization, DIMACS Series in Discrete Mathematics and Computer Science* (ed. W.J. Cook, L. Lovasz, and P. Seymour, 111–152. AMS).
- 192** Karmarkar, U.S. and Schrage, L. (1985). The deterministic dynamic product cycling problem. *Operations Research* 33: 326–345.
- 193** Karp, R.M. (1972). Reducibility among combinatorial problems. In: *Complexity of Computer Computations* (ed. R.E. Miller and J.W. Thatcher), 85–103. New York: Plenum Press.
- 194** Karp, R.M. (1975). On the complexity of combinatorial problems. *Networks* 5: 45–68.
- 195** Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Berlin: Springer-Verlag.
- 196** Khalil, E.B., Le Bodic, P., Song, L. et al. (2016). Learning to branch in mixed integer programming. *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. Phoenix, Arizona, pp. 724–731.
- 197** Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P. (1983). Optimization by simulated annealing. *Science* 220: 671–680.
- 198** Korte, B. and Vygen, J. (2005). *Combinatorial Optimization: Theory and Algorithms*, 3e. Berlin: Springer-Verlag.
- 199** Kruskal, J.B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* 7: 48–50.

- 200** Kuhn, H.W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2: 83–97.
- 201** Land, A.H. and Doig, A.G. (1960). An automatic method for solving discrete programming problems. *Econometrica* 28: 497–520.
- 202** Laporte, G. and Louveaux, F.V. (1993). The integer L-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters* 13: 133–142.
- 203** Laporte, G., Nickel, S., and da Gama, F.S. (eds.) (2015). *Location Science*. Springer.
- 204** Lawler, E.L. (1976). *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston.
- 205** Lemaréchal, C. (2001). Lagrangian relaxation. In: *Computational Combinatorial Optimization: Papers from the Spring School Held in Schloss Dagstuhl, May 15–19, 2000, Lecture Notes in Computer Science* 2241 (ed. M. Junger and D. Naddef), 112–156. Berlin: Springer-Verlag.
- 206** Lenstra, H.W. Jr. (1983). Integer programming with a fixed number of variables. *Mathematics of Operations Research* 8: 538–547.
- 207** Letchford, A.N., Eglese, R.W., and Lysgaard, J. (2002). Multistars, partial multistars and the capacitated vehicle routing problem. *Mathematical Programming* 94: 21–40.
- 208** Lin, S. and Kernighan, B.W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21 (2): 498–516.
- 209** Linderoth, J. and Savelsbergh, M.W.P. (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal of Computing* 11 (2): 173–187.
- 210** LINDO. Mathematical programming system. <https://lindo.com> (accessed 21 April 2020).
- 211** Little, J.D.C., Murty, K.G., Sweeney, D.W., and Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations Research* 11: 972–989.
- 212** Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *TOP* 25: 207–236.
- 213** Lovász, L. and Plummer, M.D. (1986). *Matching Theory*. Amsterdam: North-Holland.
- 214** Lovász, L. and Schrijver, A. (1991). Cones of matrices and set functions and 0-1 optimization. *SIAM Journal on Optimization* 1: 166–190.
- 215** Lysgaard, J. (2003). CVRPSEP: a package of separation routines for the capacitated vehicle routing problem. www.asb.dk/-lys (accessed 21 April 2020).
- 216** Lysgaard, J., Letchford, A.N., and Eglese, R.W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming* 100 (2): 423–445.

- 217 Magnanti, T.L. and Wolsey, L.A. (1995). Optimal trees. In: *Network Models, Handbooks in Operations Research and Management Science*, vol. 7 (ed. M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser), 503–615. Amsterdam: North-Holland.
- 218 Magnanti, T.L. and Wong, R.T. (1981). Accelerating Benders decomposition: algorithmic enhancement and model selection criteria. *Operations Research* 29: 464–484.
- 219 Maher, S.J. Implementing the Branch-and-Cut Approach for a general purpose Benders Decomposition Framework. *Report*. Exeter, United Kingdom: College of Engineering, Mathematics and Physical Sciences, University of Exeter.
- 220 Marchand, H. and Wolsey, L.A. (1999). The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming* 85: 15–33.
- 221 Margot, F. (2002). Pruning by isomorphism in branch-and-cut. *Mathematical Programming* 94: 71–90.
- 222 Margot, F. (2003). Exploiting orbits in symmetric ILP. *Mathematical Programming* 98: 3–21.
- 223 Margot, F. (2010). Symmetry in integer linear programming. In: *50 Years of Integer Programming 1958–2008*, Chapter 17 (ed. M. Junger, T. Liebling, D. Naddef et al.), 647–686. Heidelberg: Springer-Verlag.
- 224 Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Chichester: Wiley.
- 225 Martin, R.K. (1991). Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters* 10 (3): 119–128.
- 226 Martin, R.K. (1999). *Large Scale Linear and Integer Optimization: A Unified Approach*. Boston, MA: Kluwer Academic Publishers.
- 227 Metropolis, N., Rosenbluth, A., Rosenbluth, H. et al. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics* 21: 1087–1091.
- 228 Minkowski, H. (1896). *Geometrie der Zahlen*. Leipzig: Teubner.
- 229 Mirchandani, P.B. and Borenstein, D. (2017). A Lagrangian heuristic for the real-time vehicle rescheduling problem. *Mathematical Programming Computation* 9 (4): 573–604.
- 230 Mirchandani, P.B. and Francis, R.L. (eds.) (1990). *Discrete Location Theory*. New York: Wiley.
- 231 Morrison, D.R., Jacobson, S.H., Sauppe, J.J., and Sewell, E.C. (2016). Branch-and-bound algorithms: a survey of recent advances in searching, branching, and pruning. *Discrete Optimization* 19: 79–102.
- 232 Nemhauser, G.L. and Trotter, L.E. (1974). Properties of vertex packing and independence system polyhedra. *Mathematical Programming* 6: 48–61.

- 233** Nemhauser, G.L. and Wolsey, L.A. (1988). *Integer and Combinatorial Optimization*. New York: Wiley.
- 234** Nemhauser, G.L. and Wolsey, L.A. (1990). A recursive procedure for generating all cuts for 0-1 mixed integer programs. *Mathematical Programming* 46: 379–390.
- 235** Norman, R.Z. and Rabin, M.D. (1959). An algorithm for the minimum cover of a graph. *Proceedings of the American Mathematical Society* 10: 315–319.
- 236** Osman, I.H. and Laporte, G. (1996). Metaheuristics: a bibliography. *Annals of Operations Research* 63: 513–623.
- 237** Ostrowski, J. (2009). Symmetry in integer programming. PhD in Industrial and Systems Engineering. Lehigh University.
- 238** Ostrowski, J., Linderöth, J., Rossi, F., and Smriglio, S. (2011). Orbital branching. *Mathematical Programming* 126: 147–178.
- 239** Padberg, M.W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming* 5: 199–215.
- 240** Padberg, M. and Rao, M. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research* 7 (1): 67–80.
- 241** Padberg, M.W. and Rinaldi, G. (1991). A branch and cut algorithm for resolution of large scale symmetric traveling salesman problems. *SIAM Review* 33: 60–100.
- 242** Padberg, M.W., Van Roy, T.J., and Wolsey, L.A. (1985). Valid linear inequalities for fixed charge problems. *Operations Research* 33: 842–861.
- 243** Papadimitriou, C.H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall.
- 244** Papadimitriou, C.H. and Yannakakis, M. (1984). The complexity of facets (and some facets of complexity). *Journal of Computing and System Science* 28: 244–259.
- 245** Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation* 9: 61–100.
- 246** Pessoa, A., De Aragao, M.P., and Uchoa, E. (2008). Robust branch-cut-and-price algorithms for vehicle routing problems. In: *The Vehicle Routing Problem* (ed. B. Golden, S. Raghavan, and E. Wasil), 297–325. Springer.
- 247** Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2018). Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing* 30 (2): 339–360.
- 248** Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2019). A generic exact solver for vehicle routing and related problems. In: *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, vol. 11480, 354–369. Springer.

- 249** Pfetsch, M. and Rehn, T. (2019). A computational comparison of symmetry handling methods for mixed integer programs. *Mathematical Programming Computation* 11 (1): 37–93.
- 250** Pochet, Y. and Wolsey, L.A. (2006). *Production Planning by Mixed Integer Programming, Springer Series in Operations Research and Financial Engineering*. New York: Springer.
- 251** Prim, R.C. (1957). Shortest connection networks and some generalizations. *Bell System Technological Journal* 36: 1389–1401.
- 252** Psaraftis, H.N. (1980). A dynamic programming approach for sequencing groups of identical jobs. *Operations Research* 28: 1347–1359.
- 253** Pulleyblank, W.R. (1983). Polyhedral combinatorics. In: *Mathematical Programming: The State of the Art* (ed. A. Bachem, M., Grötschel, and B. Korte), 312–345. Berlin: Springer-Verlag.
- 254** Queyranne, M. and Schulz, A.S. (1994). *Polyhedral Approaches to Machine Scheduling*, Preprint 408/1994. Berlin: Department of Mathematics, Technical University of Berlin.
- 255** Rahamanian, R., Ahmed, S., Crainic, T.G. et al. (2018). The Benders Dual Decomposition Method. *CIRRELT-2018-03*. Université de Montréal.
- 256** Rahamanian, R., Crainic, T.G., Gendreau, M., and Rei, W. (2017). The Benders decomposition algorithm: a literature review. *European Journal of Operational Research* 259: 801–817.
- 257** Rardin, R. and Choe, U. (1979). Tighter Relaxations of Fixed Charge Network Flow Problems. *Industrial and Systems Engineering Report J-79-18*. Georgia Institute of Technology.
- 258** Rhys, J.M.W. (1970). A selection problem of shared fixed costs and network flows. *Management Science* 17: 200–207.
- 259** Rothberg, E. (2007). An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal of Computing* 19 (4): 534–541.
- 260** Rothvoss, T. (2014). The matching polytope has exponential extension complexity. *Proceedings of the 46th ACM Symposium on Theory of Computing*. New York, NY, pp. 263–272.
- 261** Ryan, D.M. and Foster, B.A. (1981). An integer programming approach to scheduling. In: *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling* (ed. A. Wren), 269–280. Amsterdam: North-Holland.
- 262** Sadykov, R., Vanderbeck, F., Pessoa, A. et al. (2019). Primal heuristics for branch-and-price: the assets of diving methods. *INFORMS Journal on Computing*, 31(2), 251–267.
- 263** Savelsbergh, M.W.P. (1993). A branch and price algorithm for the generalized assignment problem. *Operations Research* 45 (6): 831–841.

- 264** Savelsbergh, M.W.P. (1994). Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing* 6: 445–454.
- 265** Schrijver, A. (1980). On cutting planes. *Annals of Discrete Mathematics* 9: 291–296.
- 266** Schrijver, A. (1986). *Theory of Linear and Integer Programming*. Chichester: Wiley.
- 267** Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Berlin, Heidelberg: Springer.
- 268** SCIP (2012). Mathematical optimization: solving problems using SCIP and Python. <https://sur.ly/o/scipbook.readthedocs.io> (accessed 21 April 2020).
- 269** Seymour, P.D. (1980). Decomposition of regular matroids. *Journal of Combinatorial Theory B*28: 305–359.
- 270** Sherali, H. and Adams, W. (1990). A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics* 3: 411–430.
- 271** Stoer, J. and Witzgall, C. (1970). *Convexity and Optimization in Finite Dimensions*. Springer.
- 272** Sutter, A., Vanderbeck, F., and Wolsey, L.A. (1998). Optimal placement of add/drop multiplexers: heuristic and exact algorithms. *Operations Research* 46: 719–728.
- 273** Takamizawa, K., Nishizeki, T., and Saito, N. (1982). Linear time computability of combinatorial problems on series parallel graphs. *Journal of ACM* 29: 623–641.
- 274** Tang, Y., Agrawal, S., and Faenza, Y. (2018). Reinforcement learning for integer programming: learning to cut. arXiv:1810.03218v2 [cs.LG], November 2018.
- 275** Toledo, C.F.M., Arantes, M.S., Hossomi, M.Y.B. et al. (2015). A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. *Journal of Heuristics* 21: 687–717.
- 276** Toth, P. and Vigo, D. (eds.) (2014). *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, 2e. SIAM.
- 277** Valério de Carvalho, J.M. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research* 141 (2): 253–273.
- 278** Vance, P.H., Barnhart, C., Johnson, E.L., and Nemhauser, G.L. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications* 3: 111–130.
- 279** Vanderbeck, F. (1994). Decomposition and column generation for integer programs. PhD thesis. Louvain-la-Neuve, Belgium: Faculté des Sciences Appliquées, Université Catholique de Louvain.

- 280** Vanderbeck, F. (1996). Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming* 86 (3): 565–594.
- 281** Vanderbeck, F. and Wolsey, L.A. (1996). An exact algorithm for IP column generation. *Operations Research Letters* 19: 151–159.
- 282** Vanderbeck, F. and Wolsey, L.A. (2010). Reformulation and decomposition of integer programs. In: *50 Years of Integer Programming 1958–2008* (ed. M. Junger, T. Liebling, D. Naddef et al.), 431–504. Heidelberg: Springer-Verlag.
- 283** Vanderbei, R.J. (1996). *Linear Programming: Foundations and Extensions*. Boston, MA: Kluwer.
- 284** Van Roy, T.J. and Wolsey, L.A. (1987). Solving mixed 0-1 problems by automatic reformulation. *Operations Research* 35: 45–57.
- 285** Van Slyke, R.M. and Wets, R. (1969). L-Shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics* 17: 638–663.
- 286** Vazirani, V.V. (2003). *Approximation Algorithms*. Berlin: Springer-Verlag.
- 287** Wagelmans, A.P.M., van Hoesel, C.P.M., and Kolen, A.W.J. (1992). Economic lot-sizing: an $O(n \log n)$ algorithm that runs in linear time in the Wagner–Whitin case. *Operations Research* 40 (Suppl. 1): 145–156.
- 288** Wagner, H.M. and Whitin, T.M. (1958). Dynamic version of the economic lot size model. *Management Science* 5: 89–96.
- 289** Weismantel, R. (1997). On the 0/1 knapsack polytope. *Mathematical Programming* 77: 49–68.
- 290** Weninger, D. and Wolsey, L.A. (2019). Benders' algorithm with (mixed)-Integer subproblems. *Core Discussion Paper 2019–20*. Louvain-la-Neuve, Belgium: UCLouvain.
- 291** Weyl, H. (1950). The elementary theory of convex polyhedra. In: *Contributions to the Theory of Games I* (ed. H.W. Kuhn and A.W. Tucker), 3–18. Princeton, NJ: Princeton University Press.
- 292** Williamson, D.P. (2019). *Network Flow Algorithms*. New York: Cambridge University Press.
- 293** Williamson, D.P. and Shmoys, D.B. (2011). *The Design and Analysis of Approximation Algorithms*. Cambridge University Press.
- 294** Wolsey, L.A. (1975). Faces for a linear inequality in 0-1 variables. *Mathematical Programming* 8: 165–178.
- 295** Wolsey, L.A. (1975). Facets and strong valid inequalities for integer programs. *Operations Research* 24: 367–372.
- 296** Wolsey, L.A. (1977). Valid inequalities, covering problems and discrete dynamic programs. *Annals of Discrete Mathematics* 1: 527–538.
- 297** Wolsey, L.A. (1981). A resource decomposition algorithm for general mathematical programs. *Mathematical Programming Studies* 14: 244–257.

- 298** Wolsey, L.A. (1990). Valid inequalities for mixed integer programs with generalised and variable upper bound constraints. *Discrete Applied Mathematics* 25: 251–261.
- 299** Wong, R.T. (1980). Integer programming formulations of the traveling salesman problem. *Proceedings of 1980 IEEE International Conference on Circuits and Computers*. Port Chester, NY, pp. 149–152.
- 300** Wright, S. (1997). *Primal-Dual Interior Point Algorithms*. Philadelphia, PA: SIAM Publications.
- 301** XPRESS. <https://www.fico.com/en/products/fico-xpress-optimization> (accessed 21 April 2020).
- 302** Yannakakis, M. (1991). Expressing combinatorial optimization problems by linear programs. *Journal Computer and System Sciences* 43: 441–466.
- 303** Zangwill, W.I. (1966). A deterministic multi-period production scheduling model with backlogging. *Management Science* 13: 105–119.
- 304** Ziegler, G.M. (1995). *Lectures on Polytopes*. New York: Springer-Verlag.

Index

- \mathcal{NP} 97
- \mathcal{NPC} 98
- \mathcal{P} 97
- \mathcal{NP} -hard 104
- $3-SAT$ 101
- 1-tree 29, 198
- 2-partition problem 110

- a**
- Active node 117
- Affine independence 170
- Algorithm
 - primal-dual 69
- Aspiration level 256
- Assignment problem 5, 8, 67, 69, 95
 - generalized 208, 258

- b**
- Backlogging 92
- Benders' algorithm
 - integer subproblems 244
 - LP subproblem 242
 - multiple subproblems 240
 - reformulation 236
- Binary classification problem
 - 286
- Bounding 117
- Branch-and-bound 116, 123
 - estimations 122
 - LP-based 120
 - tree storage 122
- Branch-and-cut 123, 186, 284
 - Benders 237
 - performance 279
- Branch-and-price 219
 - branching rules 225
- Branch-cut-and-price 222, 226
- Branching 117
- Branching scheme 219

- c**
- Changeover costs/times 278, 280
- Chinese postman problem 77
- Chvátal-Gomory inequality
 - 145, 284
- Clique 39
- Clique finding 133
- Clustering problem 224
- co- \mathcal{NP} 111
- Column generation 213, 216, 285
- Combinatorial optimization problem 4
- Complementarity condition 196
- Complexity 71, 95
 - extended formulation 105
 - optimization and separation 104

Integer Programming, Second Edition. Laurence A. Wolsey.

© 2021 John Wiley & Sons, Inc. Published 2021 by John Wiley & Sons, Inc.

Companion website: www.wiley.com/go/wolsey/integerprogramming2e

- Consecutive 1's property 60
 - Constraint
 - aggregation 275
 - complicating 195
 - joint 203
 - redundant 130
 - Convergence 203
 - Convex function 201
 - Convex hull 15, 74, 201
 - proof 172
 - property 43, 52
 - Cover 175
 - excess 179
 - generalized 179
 - inequality 176
 - extended 176
 - separation 178
 - Cut
 - directed 53
 - minimum s-t 49
 - Cut pool 186
 - Cut-set constraint 8
 - Cutting plane algorithm 149, 280
 - Gomory fractional 150
 - Cutting stock problem 89
- d**
- D-inequality 164
 - Dantzig-Wolfe reformulation
 - convexification 215
 - discretization 215
 - see Column generation 205
 - Decision problem 96
 - Decomposition 113, 148, 168, 213
 - Benders 236
 - Lagrangian relaxation 195
 - multiple subproblems 222
 - partitioning problems 223
 - Digraph 53
 - acyclic 79
 - Dimension 170
- e**
- Discrete alternatives 11
 - Disjunction 11, 158
 - Divide and conquer 113
 - Dominance 168
 - Dual
 - Lagrangian 196, 207
 - matching 31
 - strong 30
 - superadditive 32
 - weak 30–31
 - Dual bound 26, 203
 - Duality 30, 116
 - Dynamic programming 79, 229
 - recursion 80
- f**
- Face 170
 - Facet 170
 - generation 242
 - proof 172
 - Facility location problem
 - capacitated 140
 - uncapacitated 10, 13, 16, 37, 95, 148, 197, 241, 253
 - Farkas' Lemma 33
 - Feasibility cut 237
 - Fixed costs 9
 - Flow
 - decomposition 90

- directed 53
- Flow conservation constraint 279
- Flow cover inequality 169, 275
 - separation 181
- Forest 50
- Formulation 5, 12
 - mixed integer 9
 - better 16
 - extended 14, 58
 - ideal 15
 - improved 272
 - strong 148
 - weak 148
- Fourier-Motzkin elimination 41

- g**
- Generalized transportation problem 142
- Genetic algorithm 257
- Gomory cut 286
 - fractional 151
 - mixed integer 155
- Graph
 - bipartite 65
 - connected 50
 - Eulerian 107
- Graph equipartition problem 37, 257
- Greedy algorithm 51, 54

- h**
- Heuristic 32, 251
 - 2-exchange 253
 - construction 259
 - dive-and-fix 260
 - extended formulation 264
 - feasibility pump 260
 - greedy 35, 206, 252
 - improvement 261
 - Lagrangian 205
 - large neighborhood search 263
 - local branching 261
- local search 36, 253
- nearest neighbor 253
- proximity search 261
- relax-and-fix 262, 281
- RINS 261
- worst case analysis 106

- i**
- In-out approach 242
- Incidence matrix 6
 - node-edge 31
- Incidence vector 5
- Incumbent 36, 119
- Independent set 56
- Inequality strengthening 133
- Input length 97
- Integer program 3, 95, 98
 - binary 3, 99
 - mixed 3, 141
 - 0-1 140
 - totally unimodular 95

- k**
- Knapsack problem
 - 0-1 6, 8, 13, 16, 35, 85, 95, 98, 100, 140, 175, 188
 - integer 29, 86, 106

- l**
- Labeling 65
- Lagrange multiplier 196
- Lagrangian decomposition 211
- Lagrangian dual 196, 202
 - solution complexity 207
 - strength 200
- Lagrangian relaxation 195, 283
- Lift-and-project 159
- Lifting 177

- Linear program 3, 34
 Linear programming
 dual 68
 Local optimality 36
 Logical inequalities 131
 Lot-sizing problem
 capacitated 100, 275
 constant capacity 149
 multi-item 223, 277
 uncapacitated 10, 17, 80, 95, 97, 272
- m**
 Machine learning 285
 Master problem 215
 Benders 237
 convexity constraint 216
 restricted 216
 Matching 63
 bipartite 65
 blossom inequality 73
 maximal 111
 maximum cardinality 31
 maximum weight 95
 non-bipartite 73
 perfect 75, 105
 Matroid 55
 Maximum flow problem 49, 95, 184
 Maximum weight tree problem 50, 95
 Minkowski's Theorem 33
 MIR inequality 153, 155, 275
 Mixed integer programming system
 271, 283
 Modeling language 269
 Multiplexer assignment problem 282
- n**
 Neighborhood 36
 Network flow problem
 fixed charge 58, 80, 179, 270
 minimum cost 46
 No-good cut 246
- Node
 exposed 63
 predecessor 83
 successor 83
 Node cover 31, 63
 minimum cardinality 31
 problem 101
 Node selection 118
- o**
 Odd hole 163
 Odd set edge cover 74
 Optimal subtour problem 183
 Optimality conditions 25
 Optimality cut 237
 Optimization problem 104
 Orbital branching 126
- p**
 Path
 alternating 63
 augmenting 63, 65
 shortest 79
 Performance guarantee
 a posteriori 251
 a priori 251
 Polyhedron 12
 minimal description 170
 Polynomial algorithm 97
 Polynomial reduction 98
 Preprocessing 129
 linear program 129
 bound tightening 129
 integer program 131
 Presolve
 see Preprocessing 129
 Primal bound 25, 34
 Primal heuristic 251
 Principle of optimality 80
 Priorities 126, 284

- Problem
 difficult 96
 easy 43, 96
 legitimate 96
- Production routing problem 264
- Projection 17
- Prune
 by bound 115
 by infeasibility 116
 by optimality 115
- Pseudo-costs 124
- q**
 Quadratic 0–1 problem 29, 184
- r**
 Reduced cost fixing 137
 Reduction lemma 96, 99
 Reformulation
 a priori 147
 automatic 149
 Relaxation 26, 116
 combinatorial 28
 Lagrangian 29
 linear programming 27
 Reoptimization 118
 Restriction 34
 Root of a tree 83
 Rounding 4
 integer 142
 mixed integer 142
 Run-times 280
 Running time 97
- s**
 Satisfiability problem 99
 Search
 best-node first 122
 depth-first 122
 Semi-continuous variables 126
- Separation problem 43, 100, 150
 subtour constraints 183
- Set covering problem 6, 8, 95, 206
- Shortest path problem 48, 95
 resource-constrained 213
- Simulated annealing 256, 285
- Software 269
- SOS1 125
- SOS2 125
- Special ordered sets 125
- Split cut 156
- Stable set 39, 141
- Start-up 278, 280
- Steiner tree problem 57, 95
- Stochastic Program
 2-stage with recourse 241
- Strong branching 124, 286
- Strong dual 49
 property 43
- Subgradient 202
- Subgradient algorithm 202
- Submodular
 optimization problem 54
 polyhedron 54
 rank function 55
 set function 54
- Subroutine optimization library 271
- Subtour
 elimination constraint 8
 generalized 183
 separation 183
- Subtree
 of a tree problem 83
- Surrogate Duality 212
- Symmetry breaking 126, 284
- t**
 Tabu search 255
 list 255
- Totally dual integral 56

- Totally unimodular 44, 47, 57, 184
 Tour 28
 2-optimal 253
 Traveling salesman problem 7, 9, 28, 95
 geometric 203
 prize collecting 183
 symmetric 28, 36, 98, 107, 198
 heuristic 107
 Tree 50
 Triangle inequality 107
- V**
- Valid inequality 139, 273
 disjunctive 149
 for IP 143
 for LP 143
 for Matching 144
- mixed 0–1 179
 mixed integer 153
 redundant 168
 strong 167–168
- Variable**
- fixing 130, 206
 most fractional 122
- Vehicle routing problem**
- capacitated 226
 ng-route 229
 load formulation 230
 q2-route 227
 subtour constraints 228
- W**
- Walk 107
 Weyl's theorem 41



Integer Programming: 2nd Edition

Solutions to Certain Exercises in IP Book

Chapter 1. Exercises 3, 6–8, 10, 16

Chapter 2. Exercises 1, 3, 7, 9, 11

Chapter 3. Exercises 2, 3, 8, 12, 14, 16

Chapter 4. Exercises 2, 6, 9, 12, 13

Chapter 5. Exercises 4, 5, 8, 11

Chapter 6. Exercises 1, 5, 9, 11, 12

Chapter 7. Exercises 3, 4, 10, 12

Chapter 8. Exercises 2, 4, 6, 8, 12, 17

Chapter 9. Exercises 3, 6, 16, 17

Chapter 10. Exercises 2, 4, 5, 6

Chapter 11. Exercises 1, 3

Chapter 12. Exercises 1, 2, 4

Chapter 13. Exercise 1

Chapter 14. Exercises 1, 3, 5



Solutions to Certain Exercises in Chapter 1

3. Modeling disjunctions.

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





- (ii) Show that an extended formulation for $P_1 \cup P_2$ is the set Q :

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

Solution:

- (i) $A^k y \leq b_k + M \mathbf{1} \delta_k \quad k \in [1, 2]$
 $0 \leq y \leq u$
 $\delta_1 + \delta_2 = 1$
 $\delta_k \in \{0, 1\} \quad k \in [1, 2]$
- (ii) First we show that $\text{proj}_y(Q) \subseteq P_1 \cup P_2$.
If $x^1 = 1$, then $w^2 = x^2 = 0$ leaving

$$y = w^1, \quad A^1 w^1 \leq b^1, \quad 0 \leq w^1 \leq u$$

and thus $y \in P_1$. Similarly, if $x^2 = 1$, it follows that $y \in P_2$.

Conversely, if $y \in P_1 \cup P_2$, suppose wlog that $y \in P_1$. Then $(y, w^1, w^2, x^1, x^2) \in Q$ with $w^1 = y, w^2 = 0, x^1 = 1, x^2 = 0$.

6. Prove that the set of feasible solutions to the formulation of the traveling salesman problem in Section 1.3 is precisely the set of incidence vectors of tours.

Solution: The solutions of the set

$$\left\{ x \in \mathbb{Z}_+^{n(n-1)} : \sum_{j:j \neq i} x_{ij} = 1 \quad i \in [1, n], \sum_{i:i \neq j} x_{ij} = 1 \quad j \in [1, n] \right\}$$

are assignments, namely a set of disjoint cycles, see Figure 1.2. The subtour elimination constraints eliminate any solution consisting of two or more cycles. Thus, the only solutions remaining are the 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.

Solution:

- (i) Let $x_t^i = 1$ if job i starts in period t .

Each job i must start in some period and terminate before the end of the time horizon

$$\sum_{u=1}^{9-p_i+1} x_u^i = 1 \quad i \in [1, 5].$$

If job i starts in period u , the number of workers required by job i in period t is $\ell_{i,t-u+1}$ for $t \in [u, u + p_i - 1]$. So the bound on the number of workers available in period t gives:

$$\sum_{i=1}^5 \sum_{u=t-p_i+1}^t \ell_{i,t-u+1} x_u^i \leq L_t \quad t \in [1, 9].$$

The remaining constraints are $x_t^i \in \{0, 1\}$ and $x_u^i = 0$ if $u > 9 - p_i + 1$.

- (ii) Using (i), let $\eta_t = \sum_{i=1}^5 \sum_{u=t-p_i+1}^t \ell_{i,t-u+1} x_u^i$ be the number of workers used in period t . Add the constraints $\eta_t \leq \eta$ and the objective function $\min \eta$.
- (iii) If job 1 has not started in the first t periods, job 3 cannot start in the first $t + 2$ periods.

$$\sum_{u=1}^t x_u^1 \geq \sum_{u=1}^{t+2} x_u^3 \quad t \in [1, 7].$$



- (iv) If job 5 has started in the first t periods, job 4 must have started in the first $t + 1$ periods.

$$\sum_{u=1}^t x_u^5 \leq \sum_{u=1}^{t+1} x_u^4 \quad t \in [1, 8].$$

- (v) Jobs 1 and 2 cannot both be under way in period t .

$$\sum_{u=t-p_1+1}^t x_u^1 + \sum_{u=t-p_2+1}^t x_u^2 \leq 1 \quad t \in [1, 9].$$

- 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

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

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

Solution: Suppose (x, y) is an optimal solution and let $S = \{j \in N : x_j = 1\}$.

Then, $\sum_{j \in S} f_j = \sum_{j \in N} f_j x_j$.

Also for fixed $i \in M$, as $0 \leq y_{ij} \leq x_j = 0$ for $j \notin S$, $\min\{\sum_{j \in N} c_{ij} y_{ij} : \sum_{j \in N} y_{ij} = 1, y_{i.} \in \mathbb{R}_+^n\} = \min\{\sum_{j \in S} c_{ij} y_{ij} : \sum_{j \in S} y_{ij} = 1, y_{i.} \in \mathbb{R}_+^n\} = \min_{j \in S} c_{ij}$. The result follows.

- 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.

Solution 1: Let $\delta_{ij} = 1$ if job i precedes job j and let t_j be the start-time of job j .

$$\begin{aligned} & \min \sum_{j=1}^n w_j t_j \\ & t_j \geq \sum_{i:i \neq j} p_i \delta_{ij} \\ & \delta_{ij} + \delta_{ji} = 1 \quad 1 \leq i < j \leq n \\ & \delta_{ij} + \delta_{jk} + \delta_{ki} \leq 2 \quad \forall i, j, k, i \neq j \neq k \\ & \delta_{ij} \in \{0, 1\} \quad \forall i, j, i \neq j \end{aligned} \tag{1}$$



Note that if no other constraints are present, the constraints (1) and the integrality of the δ_{ij} can be dropped. The resulting linear program has optimal solutions with $\delta_{ij} \in \{0, 1\}$ satisfying (1) in which i precedes j if $\frac{w_i}{p_i} > \frac{w_j}{p_j}$. This is commonly known as Smith's rule.

Solution 2: (Processing times p_j are integer). Let $y_{it} = 1$ if job i starts in period t .

$$\begin{aligned} \min \sum_{j=1}^n w_j (\sum_t t y_{jt}) \\ \sum_t y_{it} = 1 \quad \forall i \\ \sum_j \sum_{u=t-p_j+1}^t y_{ju} \leq 1 \quad \forall t \end{aligned} \quad (2)$$

$$y \in \{0, 1\}^{nT}$$

T is the time horizon. The constraint (2) ensures that job i is started and (3) that there is at most one job active at time t .

- 16.** Frequency Assignment. Frequencies from the range $\{1, \dots, 6\}$ must be assigned to 10 stations. For each pair of stations, there is an interference parameter which is the minimum amount by which the frequencies of the two stations must differ. The pairs with nonzero parameter are given below:

$$\begin{aligned} e = & (5, 7) (2, 3) (3, 4) (4, 5) (5, 6) (6, 7) (7, 8) (8, 9) \\ & 3 \quad 2 \quad 4 \quad 2 \quad 3 \quad 1 \quad 1 \quad 2 \\ e = & (9, 10) (1, 8) (2, 10) (3, 10) (5, 10) (7, 10) (2, 5) (5, 9) \\ & 3 \quad 2 \quad 1 \quad 1 \quad 4 \quad 2 \quad 2 \quad 2 \end{aligned}$$

The goal is to minimize the difference between the smallest and largest frequency assigned. Formulate and solve.

Solution: Let $x_i \in \mathbb{Z} \cap [1, 6]$ be the value assigned to station $i = 1, \dots, n$. Let $e = (i, j)$ with $i < j$ be the station pairs whose frequencies need to be separated by d_e . Let $z_e = 0$ if the constraint is satisfied with $x_j - x_i \geq d_e$ and $z_e = 1$ if the constraint is satisfied with $x_i - x_j \geq d_e$.

Feasibility IP:

$$\begin{aligned} x_j - x_i &\geq d_e - Mz_e \quad \forall e \in E \\ x_i - x_j &\geq d_e - M(1 - z_e) \quad \forall e \in E \\ 1 \leq x_j &\leq 6 \quad \forall j \\ x \in \mathbb{Z}^n, z &\in \{0, 1\}^{|E|}. \end{aligned}$$



Solutions to Certain Exercises in Chapter 2

1. Find a maximum cardinality matching in the graph of Figure 2.5a by inspection. Give a proof that the solution found is optimal.

Solution: Matching $M = \{(2, 2'), (3, 1'), (4, 3'), (5, 5'), (6, 6')\}$ and node cover $R = \{2, 3, 6, 3', 5'\}$ with $|M| = |R|$.

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

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

Solution: As $\frac{c_1}{a_1} > \frac{c_j}{a_j}$ for $j > 1$, the solution of the linear programming relaxation is $x_1 = \frac{69}{14} = 4\frac{13}{14}$ giving an upper bound of $\frac{69}{14} \cdot 42 = 207$. A first greedy heuristic solution is $x_1 = \lfloor \frac{69}{14} \rfloor = 4$ giving a lower bound of 168. With $x_1 = 4$, there remain $69 - 56 = 13$ units available in the knapsack. This leads to an improved greedy solution of $x_1 = 4, x_3 = 1$ of value 203.

7. Network Formulation of the Integer Knapsack Problem with positive integer coefficients $a_j > 0$ for $j \in [1, n]$ and $b > 0$.

Let

$$G(d) = \max \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x \in Z_+^n \right\}$$

denote problem $P(d)$.

- (i) Show that $G(d) \geq G(d - a_j) + c_j \forall a_j \leq d \leq b$ and $j \in [1, n]$ and $G(d) \geq G(d - 1) \forall 1 \leq d \leq b$.
(ii) Show that

$$\begin{aligned} \min y(b) \\ y(d) - y(d - a_j) \geq c_j \forall j, d \\ y(d) - y(d - 1) \geq 0 \forall d \\ y(0) = 0 \end{aligned}$$

is a strong dual of the knapsack problem.

- (iii) Take the dual of this linear program and interpret the variables and constraints of the resulting LP.

Solution:

- (i) Let x^* be an optimal solution to problem $P(d - a_j)$ with $G(d - a_j) = cx^*$ and $Ax^* \leq d - a_j$. Then $x^* + e_j$ is feasible in $P(d)$ with value $cx^* + c_j$. So $G(d) \geq cx^* + c_j = G(d - a_j) + c_j$.
 Let x^* be an optimal solution to problem $P(d - 1)$ with $G(d - 1) = cx^*$ and $Ax^* \leq d - 1$. Then, x^* is feasible in $P(d)$ with value cx^* . So $G(d) \geq G(d - 1)$.
- (ii) Take $y(d) = G(d)$ for $d \in [0, b]$. Then, $G(b)$ is both an upper bound on the optimal value and the value of a primal feasible solution.
- (iii) The dual is

$$\begin{aligned} & \max \sum_{d,j} c_j x_{j,d} \\ & \sum_j x_{j,d} - \sum_j x_{j,d+a_j} + z_d - z_{d+1} = 0 \quad 0 \leq d < b \\ & \sum_j x_{j,b} + z_b = 1 \\ & x_{j,d} \in \mathbb{R}_+^1, z_d \in \mathbb{R}_+^1 \end{aligned}$$

This is a longest path problem in an acyclic graph with nodes $V = \{0, 1, \dots, b\}$, $x_{j,d} = 1$ if the arc $(d - a_j, d)$ is used and $z_d = 1$ if the arc $(d - 1, d)$ is used.



9. Suppose that the value function $\phi(d) = \max\{cx : Ax \leq d, x \in \mathbb{Z}_+^n\}$ is finite valued for all $d \in \mathbb{Z}^m$.

- (i) Show that the value function ϕ is superadditive.
 (ii) Prove the Strong Duality Theorem 2.8.

Solution:

- (i) $\phi(0) \geq 0$. If $\phi(0) > 0$, $\phi(d)$ is unbounded, a contradiction. Thus $\phi(0) = 0$.
 Suppose that $\phi(d^i) = cx^i$ with $Ax^i \leq d^i, x^i \in \mathbb{Z}_+^n$ for $i = 1, 2$. Then $x^1 + x^2$ satisfies $A(x^1 + x^2) \leq d^1 + d^2, x^1 + x^2 \in \mathbb{Z}_+^n$ and has value $c(x^1 + x^2)$. It follows that $\phi(d^1 + d^2) \geq c(x^1 + x^2) = \phi(d^1) + \phi(d^2)$. Showing that ϕ is nondecreasing is similar.
- (ii) $\phi(b)$ is a feasible value for both the primal and the dual as $\phi(a_j) \geq c_j$ for $j \in [1, n]$.





11. *Fourier–Motzkin elimination.* Let Q be the polyhedron

$$a^i y + w \leq b_i \quad i \in [1, m] \quad (4)$$

$$c^j y - w \leq d_j \quad j \in [1, n] \quad (5)$$

$$e^k y \leq f_k \quad k \in [1, K] \quad (6)$$

$$y \in \mathbb{R}^p, w \in \mathbb{R}^1. \quad (7)$$

Show that $P = \text{proj}_y(Q)$, where P is the polyhedron

$$(a^i + c^j)y \leq b_i + d_j \quad i \in [1, m], j \in [1, n] \quad (8)$$

$$e^k y \leq f_k \quad k \in [1, K] \quad (9)$$

$$y \in \mathbb{R}^p. \quad (10)$$

Repeating this procedure allows one to project any polyhedron onto a subset of its variables. Note that the number of constraints in the projected polyhedron can increase exponentially.

Solution: If $y \in \text{proj}_y(Q)$, there exists w such that $(y, w) \in Q$. Combining the valid inequalities (4) and (5) gives (8) and thus $y \in P$. So, $\text{proj}_y(Q) \subseteq P$. Conversely, suppose that $y \in P$. From (8), it follows that $c^j y - d_j \leq b_i - a^i y$ for all i, j and thus $\max_j [c^j y - d_j] \leq \min_i [b_i - a^i y]$. Taking w such that $\max_j [c^j y - d_j] \leq w \leq \min_i [b_i - a^i y]$, $w \leq b_i - a^i y$ for all i and $w \geq c^j y - d_j$ for all j and thus $(y, w) \in Q$. So, $P \subseteq \text{proj}_y(Q)$.



Solutions to Certain Exercises in Chapter 3

- 2.** Prove that the polyhedron $P = \{(y_1, \dots, y_m, x) \in R_+^{m+1} : x \leq 1, y_i \leq x \text{ for } i = 1, \dots, m\}$ has integer vertices.

Solution: The matrix A obtained from the constraints $x_i - y \leq 0$ $i \in [1, m]$ has a $+1$ and a -1 in each row. So it is transpose of a network matrix. Network matrices are TU by Proposition 3.4. The dual of a network matrix is TU by Proposition 3.1. As the rhs and bound vectors are integers, the vertices of P are integral by Proposition 3.3.

- 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.

Solution: Take any subset of the rows M . The matrix A^* obtained by removing the rows not in M still has the consecutive 1's property. Assign the remaining rows so that odd rows belong to M_1 and even rows to M_2 . Then for column j , if the first 1 appears in an odd numbered row, $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} \in \{0, 1\}$ and if the first 1 appears in an even numbered row, $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} \in \{0, -1\}$.

8. Find a minimum weight spanning tree in the graph shown in Figure 3.4a.

Solution: The greedy algorithm selects the edges in the following order:

edge	cost	comment
$e_1 = (3, 6)$	1	
$e_2 = (2, 3)$	2	
$e_3 = (2, 5)$	3	
$e_4 = (5, 9)$	4	
$e_5 = (6, 9)$	—	Cycle, Reject
$e_6 = (1, 2)$	4	
$e_7 = (5, 8)$	6	
$e_8 = (3, 7)$	6	
$e_9 = (1, 6)$	—	Cycle. Reject
$e_{10} = (4, 8)$	8	Finish.

12. Show that the 0–1 covering problem

$$\begin{aligned} z &= \min \sum_{j=1}^n f_j x_j \\ \sum_{j=1}^n a_{ij} x_j &\geq b_i \quad \text{for } i \in M \\ x &\in \{0, 1\}^n \end{aligned}$$

with $a_{ij} \geq 0$ for $i \in M, j \in N$ can be written in the form

$$z = \min \left\{ \sum_{j \in S} f_j : g(S) = g(N) \right\},$$

where $g : \mathcal{P}(N) \rightarrow \mathbb{R}_+^1$ is a nondecreasing set function.

- (i) What is g ?
(ii) Show that g is submodular and nondecreasing.

Solution:

- (i) $g(S) = \sum_{i \in M} \min(\sum_{j \in S} a_{ij}, b_i)$.
(ii) Suppose wlog that $|M| = 1$. $\rho_j(S) = g(S \cup j) - g(S)$.
As g is nondecreasing, ρ_j is nonnegative.
Case 1. If $g(S \cup k) \geq b$, $\rho_j(S \cup k) = 0 \leq \rho_j(S)$.
Case 2. If $g(S \cup k) < b$, then $\rho_j(S \cup k) = \min[b - \sum_{i \in S \cup k} a_i, a_j]$. As this also implies $g(S) < b$, $\rho_j(S) = \min[b - \sum_{i \in S} a_i, a_j] \geq \rho_j(S \cup k)$. The claim follows.

- 14.** 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 $B \cup \{j\}$ is independent, and
- (ii) for an arbitrary set $A \subseteq N$, every maximal independent set in A has the same cardinality.

Solution:

- (i) Let r be the rank function of the matroid. As A and B are independent, $r(A) = |A|$ and $r(B) = |B|$ and by hypothesis $r(A) > r(B)$. Now suppose there is no $j \in A \setminus B$ such that $B \cup \{j\}$ is independent. In other words, $r(B) = r(B \cup \{j\})$ for all $j \in A \setminus B$. By Proposition 3.11,

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

giving $r(A) \leq r(B) + \sum_{j \in A \setminus B} 0 = r(B)$, a contradiction.

- (ii) follows from (i).

- 16.** Consider the Steiner tree problem on the graph of Figure 3.5 with terminal nodes 0, 4, 5, 6 and a weight of 1 on all the edges. Show that the LP relaxation of the directed cut formulation (3.15)–(3.17) is not integral by describing a fractional LP solution of smaller value than that of the optimal integer solution.

Solution: Observe that all the directed cuts contain at least two arcs. Setting $z_{ij} = \frac{1}{2}$ on each downward directed arc provides a feasible solution with $x_e = \frac{1}{2}$ on each edge and value $\frac{9}{2}$. Clearly, the optimal integer solution is 5 as there is no solution using only one of the edges adjacent to node 0 and at least 3 of the other edges are needed.

Solutions to Certain Exercises in Chapter 4

2. Find a maximum cardinality matching in the bipartite graph of Figure 4.6b. Demonstrate that your solution is optimal.

Solution: $M = \{(1, 9), (2, 8), (3, 7), (6, 11)\}$. Augmenting path $(5, 11), (6, 11), (6, 12)$. $M = \{(1, 9), (2, 8), (3, 7), (5, 11), (6, 12)\}$. $R = \{2, 3, 6, 9, 11\}$.

6. Find a maximum weight matching in the weighted bipartite graph of Figure 4.7.

Solution: Adding a node 12 or 6', the profit matrix is

$$\begin{pmatrix} 3 & 0 & 7 & 0 & 0 & 0 \\ 8 & 0 & 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 8 & 4 & 0 \\ 8 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & 2 & 6 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 \end{pmatrix}$$

Choosing a dual solution greedily so as to have a reduced cost of zero in each row and column

$$u^0 = (7, 8, 8, 8, 6, 2), v^0 = (0, -2, 0, 0, 0, -2)$$

$$\bar{z} = 35$$

$$M = \{(1, 3'), (2, 1'), (5, 4'), (6, 6')\}$$

$$V_1^+ = \{2, 3, 4, 5\}, V_2^+ = \{1', 4'\}, \delta = 1$$

$$u^1 = (7, 7, 7, 7, 5, 2), v^1 = (1, -2, 0, 1, 0, -2)$$

$$\bar{z} = 33$$

$$M = \{(1, 3'), (2, 1'), (3, 4'), (5, 2'), (6, 6')\}$$

$$V_1^+ = \{2, 4\}, V_2^+ = \{1'\}, \delta = 3$$

$$u^2 = (7, 4, 7, 4, 5, 2), v^2 = (4, -2, 0, 1, 0, -2)$$

$$\bar{z} = 30$$

M unchanged

$$V_1^+ = \{1, 2, 4\}, V_2^+ = \{1', 3'\}, \delta = 2$$

$$u^3 = (5, 2, 7, 2, 5, 2), v^3 = (6, -2, 2, 1, 0, -2)$$

$$\bar{z} = 28$$

$$M = \{(1, 3'), (2, 1'), (3, 4'), (4, 6'), (5, 2'), (6, 5')\}$$

Optimal Solution $z = 28$.

9. 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 $S_1 = \{3, 7, 8, 10\}$, $S_2 = \{4, 8\}$, $S_3 = \{2, 5, 7\}$, $S_4 = \{1, 2, 7, 9\}$, $S_5 = \{2, 5, 7\}$, $S_6 = \{1, 4, 5, 7\}$, $S_7 = \{2, 7\}$, $S_8 = \{1, 6, 7, 10\}$, $S_9 = \{2, 5\}$, $S_{10} = \{1, 2, 3, 6, 7, 8, 10\}$?

Solution: This is known as a Systems of Discrete Representatives. Solve a maximum cardinality or max flow problem.

Bipartite graph G with $V_1 = V_2 = \{1, \dots, n\}$. $(ij) \in E$ if $j \in S_i$.

$$\begin{aligned} \max \quad & \sum_{(ij) \in E} x_{ij} \\ \text{subject to} \quad & \sum_{j:(ij) \in E} x_{ij} \leq 1 \quad i = 1, \dots, n \\ & \sum_{i:(ij) \in E} x_{ij} \leq 1 \quad j = 1, \dots, n \\ & x_{ij} \in \{0, 1\} \quad (ij) \in E. \end{aligned}$$

A primal solution: (1,10), (2,4), (3,2) (4,9) (6,1) (7,7) (8,6) (9,5) (10,8).

A dual solution: $u_i = 1$ for $i \in \{1, 2, 4, 6, 8, 10\}$, $v_j = 1$ for $j \in \{2, 5, 7\}$.

Flow = 9.

There is no such system. There are only three researchers ($v_j = 1$) available to represent the four projects ($u_i = 0$).



12. Show that the convex hull of perfect matchings in a graph on an even number of nodes is described by the degree constraints (4.1), the nonnegativity constraints (4.2) and the inequalities

$$\sum_{e \in \delta(S)} x_e \geq 1 \quad \text{for } S \text{ odd, } |S| \geq 3.$$

Solution: Let P' denote the above polytope. From Theorem 4.2, we obtain the perfect matching polytope F by adding the valid inequality $\sum_{e \in E} x_e \leq \frac{|V|}{2}$ as an equality. This in turn implies that $\sum_{e \in \delta(i)} x_e = 1$ for $i \in V$.

Now observe that the inequalities of P' are clearly satisfied by the perfect matchings and thus $F \subseteq P'$. It remains to show that $P' \subseteq F$.

Summing the degree constraints over an odd set S gives $\sum_{i \in S} \sum_{e \in \delta(i)} x_e = |S|$, or $2 \sum_{e \in E(S)} x_e + \sum_{e \in \delta(S, V \setminus S)} x_e = |S|$. Subtracting $\sum_{e \in \delta(S, V \setminus S)} x_e \geq 1$ and dividing by 2 gives $\sum_{e \in E(S)} x_e \leq \frac{|S|-1}{2}$ for S odd. As the degree and nonnegativity constraints are the same in P' and F , the claim follows.

13. (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, he 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?

Solution:

- (i) Construct a graph with one node per person. Add an edge if staff members i and j are compatible. The problem is then to find a perfect matching with 15 edges.
- (ii) Construct a bipartite graph with nodes V_1 for the English children and V_2 for the French with $|V_1| = |V_2|$. Assign weights to each pair (i,j) with $i \in V_1, j \in V_2$. The problem is to find a maximum weight assignment/matching in the bipartite graph.
- (iii) As we have seen, a maximum weight assignment can be found by linear programming. On the other hand, there is no small LP guaranteeing a solution to the perfect matching problem in an arbitrary graph. There is however an efficient algorithm. See discussion in Section 4.4.



Solutions to Certain Exercises in Chapter 5

4. Formulate the optimal subtree of a tree problem as an integer program. Is this IP easy to solve?

Solution: Let r be the root of the tree and let $p(v)$ denote the predecessor of v on the path from v to the root. Let $x_v = 1$ if node v is part of the subtree. Consider the formulation

$$\begin{aligned} \max_{v \in V} c_v x_v \\ x_r \leq 1 \\ x_v - x_{p(v)} \leq 0 \quad v \in V \setminus \{r\} \\ x \in \mathbb{R}_+^{|V|} \end{aligned}$$

Note that the constraint matrix is the dual of a network matrix. So it is TU. As the rhs vector is integer, the extreme points are integral.



5. Given a digraph $D = (V, A)$, travel times c_{ij} for $(i, j) \in A$ for traversing the arcs, and earliest passage times r_j for $j \in V$, consider the problem of minimizing the time required to go from node 1 to node n .
- Describe a dynamic programming recursion.
 - Formulate as a mixed integer program. Is this mixed integer program easy to solve?

Solution:

- (i) Let $G(j)$ be the earliest passage time at node j . Consider the recursion

$$G(j) \geq \max(r_j, \min_{i \in V \setminus \{j\}} (G(i) + c_{ij}))$$

with initialization $G(1) = r_1$.

- (ii) One possible formulation. Let t_j be the passage time at node j and $x_{ij} = 1$ if arc (i, j) lies on the path.

$$\begin{aligned} & \text{min } t_n \\ & t_j - t_i \geq c_{ij}x_{ij} - M(1 - x_{ij}) \quad i, j \in V, \quad i \neq j \\ & t_j \geq r_j \quad j \in V \\ & x \in \{0, 1\}^{|A|}. \end{aligned}$$



Because of the big M in the formulation, the linear programming relaxation will be weak.

8. Solve the problem

$$\begin{aligned} & \max x_1^3 + 2x_2^{\frac{1}{2}} + 4x_3 + 4x_4 \\ & 2x_1^2 + 4x_2 + 6x_3 + 5x_4 \leq t \\ & x_1 \leq 3, x_2 \leq 2, x_4 \leq 1 \\ & x \in \mathbb{Z}_+^4 \end{aligned}$$

for $t \leq 12$. (Hint. One of the recursions in the chapter is easily adapted.)

Solution: The problem is of the form

$$\begin{aligned} f_r(\lambda) &= \max \sum_{j=1}^n c_j(x_j) \\ & \sum_{j=1}^n a_j(x_j) \leq \lambda \\ & x_j \in [0, h_j] \cap \mathbb{Z} \quad j \in [1, n] \end{aligned}$$



Use the recursion

$$f_r(\lambda) = \max_{x_r \in [0, h_r] \cap \mathbb{Z}} (f_{r-1}(\lambda - a_r(x_r)) + c_r(x_r)).$$

11. 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.

Solution: Arbitrarily root the tree, so that each subtree is now rooted. Let $S(v)$ be the successor nodes of v in T and $\Sigma(R)$ the successor nodes of tree R . Let Ω_v be the subtrees in the set that are rooted at node v and $G(v)$ be the value of an optimal packing of the subtrees completely lying in the subtree of T rooted at v . Then, an optimal solution of value $G(v)$ either does not cover node v or it is covered by a tree R in Ω_v .

$$G(v) = \max \left(\max_{R \in \Omega_v} [c(R) + \sum_{u \in S(R)} G(u)], \sum_{u \in S(v)} G(u) \right).$$

Solutions to Certain Exercises in Chapter 6

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 $\sum_{j \in S} a_j = \sum_{j \in N \setminus S} a_j$, or prove that it is impossible. Show that 2-PARTITION is polynomially reducible to 0-1 KNAPSACK. Does this imply that 2-PARTITION is NP-complete?

Solution: 2-PARTITION is a special case of 0-1 KNAPSACK, so 2-PARTITION is polynomially reducible to 0-1 KNAPSACK. The fact that 0-1 KNAPSACK $\in \text{NP-PC}$ tells us nothing about the complexity of 2-PARTITION.

5. Show that SET COVERING is polynomially reducible to UFL.

Solution: Writing SET-COVERING as the problem

$$\min \{cx : Ax \geq 1, x \in \mathbb{Z}_+^n\}$$

with A an m by n 0-1 matrix and UFL as the problem

$$\min \left\{ cx + dy : \sum_j y_{ij} = 1 \text{ for } \forall i, y_{ij} \leq x_j \forall i, j, y \in [0, 1]^{mn}, x \in \{0, 1\}^n \right\}.$$



Take $d_{ij} = M(1 - a_{ij})$. Then, a solution of *UFL* will have $y_{ij} = 0$ whenever $a_{ij} = 0$ because of the large penalty and also satisfy $Ax \geq \mathbf{1}$ by combining $\sum_j y_{ij} = 1$ for $i = 1$ with $y_{ij} \leq x_j$.

Conversely, if x^* is optimal in *SET-COVERING*, then the solution (x^*, y^*) is feasible in *UFL* by taking $y_{i,j_i} = 1$ where $j_i = \min\{j : x^j = 1, a_{ij} = 1\}$ and has the same cost.

9. Consider a 0–1 knapsack set $X = \{x \in \{0, 1\}^n : \sum_{j \in N} a_j x_j \leq b\}$ with $0 \leq a_j \leq b$ for $j \in N$ and let $\{x^t\}_{t=1}^T$ be the points of X . With it, associate the bounded polyhedron $\Pi^1 = \{\pi \in \mathbb{R}_+^n : x^t \pi \leq 1 \text{ for } t = 1, \dots, T\}$ with extreme points $\{\pi^s\}_{s=1}^S$. Consider a point x^* with $0 \leq x_j^* \leq 1$ for $j \in N$.
 - (i) Show that $x^* \in \text{conv}(X)$ if and only if $\min\{\sum_{t=1}^T \lambda_t : x^* \leq \sum_{t=1}^T x^t \lambda_t, \lambda \in \mathbb{R}_+^T\} = \max\{x^* \pi : \pi \in \Pi^1\} \leq 1$.
 - (ii) Deduce that if $x^* \notin \text{conv}(X)$, then for some $s = 1, \dots, S$, $\pi^s x^* > 1$.

Solution:

- (i) If $x^* \in \text{conv}(X)$, then $x^* = \sum_{t=1}^T \lambda_t^* x^t$ with $\sum_{t=1}^T \lambda_t^* = 1, \lambda^* \in \mathbb{R}_+^T$. So λ^* is feasible in the LP $Z = \min \left\{ \sum_{t=1}^T \lambda_t : x^* \leq \sum_{t=1}^T \lambda_t x^t, \lambda \in \mathbb{R}_+^T \right\}$ with objective value 1 and so $Z \leq 1$. Conversely if $Z = \min \left\{ \sum_{t=1}^T \lambda_t : x^* \leq \sum_{t=1}^T \lambda_t x^t, \lambda \in \mathbb{R}_+^T \right\} \leq 1$, let $\hat{\lambda}$ be an optimal solution. The $x^* = \sum_{t=1}^T \hat{\lambda}_t x^t + (1 - Z)\mathbf{0} \in \text{conv}(X)$. Just take the LP dual.
- (ii) Follows from if and only if.

11. Consider the 0–1 knapsack problem: $Z = \max\{\sum_{j \in N} c_j x_j : \sum_{j \in N} a_j x_j \leq b, x \in \{0, 1\}^n\}$ with $0 < a_j \leq b$ for $j \in N$. Consider a greedy heuristic that chooses the better of the integer round down of the linear programming solution, and the best solution in which just one variable $x_k = 1$ where $c_k = \max_j c_j$. Show that $z^G \geq \frac{1}{2}Z$.

Solution: Suppose that the items are ordered so that $\frac{c_1}{a_1} \geq \dots \geq \frac{c_n}{a_n}$. The linear programming solution is then $x_1 = \dots = x_{r-1} = 1, x_r = \frac{b - \sum_{j=1}^{r-1} a_j}{a_r}$ for some r with value Z^{LP} . The greedy solution is $x_1 = \dots = x_{r-1} = 1$. Otherwise set $x_r = 1$.

$$z^G = \max \left(\sum_{j=1}^{r-1} c_j, c_r \right) \geq \max \left(\sum_{j=1}^{r-1} c_j, c_r \right) \geq \frac{1}{2} \left(\sum_{j=1}^{r-1} c_j + c_r \right) \geq \frac{1}{2} Z^{\text{LP}} \geq \frac{1}{2} Z$$

as required.



12. Consider the problem of finding a maximum cardinality matching from Chapter 4. A matching $M \subseteq E$ is *maximal* if $M \cup \{f\}$ is not a matching for any $f \in E \setminus M$. Let Z be the size of a maximum matching. Show that $|M| \geq \frac{1}{2}Z$ for any maximal matching M .

Solution: Consider the dual linear program: $z^{\text{LP}} = \min\{\sum_i u_i : u_i + u_j \geq 1$ for $(i, j) \in E, u \in \mathbb{R}_+^{|V|}\}$. Consider a maximal matching M . Set $u_i = 1$ if node i is incident to M . We claim that u is dual feasible. If not, there is some edge (i, j) with $u_i + u_j < 1$. This implies that neither node i nor node j touches M and thus M is not maximal. Now $\sum_{i \in V} u_i = 2|M| \geq Z^{\text{LP}} \geq Z$.

Solutions to Certain Exercises in Chapter 7

3. Consider the 0–1 knapsack problem:

$$\max \left\{ \sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b, x \in \{0, 1\}^n \right\}$$

with $a_j, c_j > 0$ for $j = 1, \dots, n$.

- (i) Show that if $\frac{c_1}{a_1} \geq \dots \geq \frac{c_n}{a_n} > 0$, $\sum_{j=1}^{r-1} a_j \leq b$ and $\sum_{j=1}^r a_j > b$, the solution of the LP relaxation is $x_j = 1$ for $j = 1, \dots, r-1$, $x_r = \left(b - \sum_{j=1}^{r-1} a_j\right)/a_r$, and $x_j = 0$ for $j > r$.
- (ii) Solve the instance

$$\begin{aligned} \max \quad & 17x_1 + 10x_2 + 25x_3 + 17x_4 \\ \text{s.t.} \quad & 5x_1 + 3x_2 + 8x_3 + 7x_4 \leq 12 \\ & x \in \{0, 1\}^4 \end{aligned}$$

by branch-and-bound.

Solution:

- (i) Let $\bar{x}_j = 1 - x_j$. The constraint can now be rewritten as $a_r x_r - \sum_{j=1}^{r-1} a_j \bar{x}_j + \sum_{j=r+1}^n a_j x_j = b - \sum_{j=1}^{r-1} a_j$ and the objective function as $\max c_r x_r - \sum_{j=1}^{r-1} c_j \bar{x}_j + \sum_{j=r+1}^n c_j x_j + \sum_{j=1}^{r-1} c_j$. Substituting for x_r in the objective function gives $\max \sum_{j=1}^{r-1} c_j + c_r \frac{b - \sum_{j=1}^{r-1} a_j}{a_r} + \sum_{j=1}^{r-1} (c_j - c_r \frac{a_j}{a_r}) \bar{x}_j - \sum_{j=r+1}^n (c_j - c_r \frac{a_j}{a_r}) x_j$. As all the reduced costs are nonpositive, the solution with x_r basic and values $x_r = \frac{b - \sum_{j=1}^{r-1} a_j}{a_r}$, $x_j = 1$ for $j = 1, \dots, r-1$ is optimal.

- (ii) Let b' be the amount remaining from b when fixing and lower bounds on variables are taken into account.

At any node, if $a_j > b'$, we set $x_j = 0$,
Note that $\frac{c_1}{a_1} > \dots > \frac{c_4}{a_4}$.

Node 1. LP solution $x = (1, 1, 0.5, 0)$, $Z_{LP} = 39.5$

Branch on x_3 .

Create nodes 2 and 3 with $x_3 = 0$ and $x_3 = 1$, respectively.

Node 2. $x_3 = 0$.

LP solution $x = (1, 1, 0, \frac{4}{7})$, $Z_{LP} = 36\frac{5}{7}$
Branch on x_4 .

Create nodes 4 and 5 with $x_4 = 0$ and $x_4 = 1$, respectively.

Node 4. LP solution $x = (1, 1, 0, 0)$, $Z_{LP} = 27$. New incumbent $\underline{Z} = 27$.

Prune by optimality.

Node 5. $x_4 = 1$ implies remaining $b' = 5$, so $x_3 = 0$.

LP solution $x = (1, 0, 0, 1)$, $Z_{LP} = 34$. New incumbent $\underline{Z} = 34$. Prune by optimality.

Node 3. $x_3 = 1$ implies $x_1 = x_4 = 0$.

LP Solution $x = (0, 1, 1, 0)$, $Z_{LP} = 35$. New incumbent $\underline{Z} = 35$. Prune by optimality.

All nodes have been pruned.



4. Solve the integer knapsack problem:

$$\begin{aligned} \max \quad & 10x_1 + 12x_2 + 7x_3 + 2x_4 \\ \text{subject to} \quad & 4x_1 + 5x_2 + 3x_3 + 1x_4 \leq 10 \\ & x_1, x_2 \in Z_+^1, x_3, x_4 \in \{0, 1\} \end{aligned}$$

by branch-and-bound.

Solution: At any node, we set $x_j \leq \lfloor \frac{b'}{a_j} \rfloor$.

Note that $\frac{c_1}{a_1} > \dots > \frac{c_4}{a_4}$.

This gives $x_1 \leq 2$ and $x_2 \leq 2$.

Node 1. LP solution $x = (2, \frac{2}{5}, 0, 0)$, $Z_{LP} = 24.8$.

As the objective function value must be integer. $Z \leq \lfloor 24.8 \rfloor = 24$.

Branch on x_2 .

Create nodes 2 and 3 with $x_2 = 0$ and $x_2 \geq 1$, respectively.

Node 3. $x_2 \geq 1$ implies $b' = 5$ and thus $x_1 \leq 1$

LP solution $x = (1, \frac{6}{5}, 0, 0)$, $Z_{LP} = 24.4$.

Branch on x_2 .

Create nodes 4 and 5 with $x_2 \leq 1$ and $x_2 \geq 2$, respectively.

Node 5. As $x_2 = 2$, $b' = 0$ and thus $x_1 = x_3 = x_4 = 0$.



LP solution $x = (0, 2, 0, 0)$, $Z_{LP} = 24$. New incumbent. $\underline{Z} = 24$.
 As lower and upper bounds on Z are equal, the incumbent is optimal.

- 10.** Prove Proposition 7.6 concerning 0–1 preprocessing.

Solution:

- (i) $X = \emptyset$ if and only if $b < 0$.
- (ii) The constraint is redundant if and only if $\sum_{j=1}^n a_j \leq b$.
- (iii) $x_j = 0$ if and only if $a_j > b$.
- (iv) $x_i + x_j \leq 1$ is valid if and only if $a_i + a_j > b$.

$$X = \{x \in \{0, 1\}^5 : x_1 + 3x_2 - 5x_3 + x_4 + 5x_5 \leq 0\}.$$

Let $\bar{x}_j = 1 - x_j$. The constraint can be rewritten as

$$x_1 + 3x_2 + 5\bar{x}_3 + x_4 + 5x_5 \leq 5.$$

From (iv), we have that $x_i + x_5 \leq 1$ whenever $i \in \{1, 2, 4\}$ and $x_i + \bar{x}_3 \leq 1$ (or $x_i \leq x_3$) for $i \in \{1, 2, 4, 5\}$.

- 12.** Suppose that $P^i = \{x \in \mathbb{R}^n : A^i x \leq b^i\}$ for $i = 1, \dots, m$ and that $C_k \subseteq \{1, \dots, m\}$ for $k = 1, \dots, K$. A *disjunctive program* is a problem of the form

$$\max\{cx : x \in \bigcup_{i \in C_k} P^i \text{ for } k = 1, \dots, K\}.$$

Show how the following can be modeled as disjunctive programs:

- (i) A 0–1 integer program.
- (ii) A linear complementarity problem: $w = q + Mz$, $w, z \in \mathbb{R}_+^m$, $w_j z_j = 0$ for $j = 1, \dots, m$.
- (iii) A single machine sequencing problem with job processing times p_j , and variables t_j representing the start time of job j for $j = 1, \dots, n$.
- (iv) The nonlinear expression $z \geq \min\{3x_1 + 2x_2 - 3, 9x_1 - 4x_2 + 6\}$.
- (v) The constraint: if $x \in \mathbb{R}_+^1$ is positive, then x lies between 20 and 50, and is a multiple of 5.

Solution:

- (i) 0–1 IP. $X \subseteq \{0, 1\}^n$.

$$\bigcup_{S \subseteq N} P^S$$

where $P^S = \{x \in X : x = x^S\}$.

- (ii) $C = \{(w, z) \in \mathbb{R}_+^m \times \mathbb{R}_+^m : w = q + Mz\}$.

$$\bigcup_{S \subseteq \{1, \dots, m\}} P^S$$

where $P^S = \{(w, z) \in C : w_i = 0 \text{ for } i \in S, z_i = 0 \text{ for } i \in \{1, \dots, m\} \setminus S\}$.

- (iii) A permutation $\pi : \{1, \dots, n\}$ gives the ordering of the jobs. $\pi(1)$ precedes $\pi(2)$, etc.

$$\cup_{\pi} P^{\pi}$$

where $P^{\pi} = \{t \in \mathbb{R}_+^n : t_{\pi(i+1)} \geq t_{\pi(i)} + p_{\pi(i)} \text{ for } i = 1, \dots, n-1\}$.

- (iv) $\{(x, z) : z \geq 3x_1 + 2x_2 - 3\} \cup \{(x, z) : z \geq 9x_1 - 4x_2 + 6\}$.
- (v) $\{x : x = 0\} \cup \bigcup_{k=4}^{10} \{(x, k) : x = 5k\}$.

Solutions to Certain Exercises in Chapter 8

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) $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^2 : y_1 + y_2 \leq 2x, y_j \leq 1 \text{ for } j = 1, 2\}$

$(x, y_1, y_2) = (0.5, 1, 0)$

(ii) $X = \{(x, y) \in \mathbb{Z}_+^1 \times \mathbb{R}_+^1 : y \leq 9, y \leq 4x\}$

$(x, y) = \left(\frac{9}{4}, 9\right)$

(iii) $X = \{(x, y_1, y_2) \in \mathbb{Z}_+^1 \times \mathbb{R}_+^2 : y_1 + y_2 \leq 25, y_1 + y_2 \leq 8x\}$

$(x, y_1, y_2) = \left(\frac{25}{8}, 20, 5\right)$

(iv) $X = \{x \in \mathbb{Z}_+^5 : 9x_1 + 12x_2 + 8x_3 + 17x_4 + 13x_5 \geq 50\}$

$x = \left(0, \frac{25}{6}, 0, 0, 0\right)$

(v) $X = \{x \in \mathbb{Z}_+^4 : 4x_1 + 8x_2 + 7x_3 + 5x_4 \leq 33\}$

$x = \left(0, 0, \frac{33}{7}, 0\right)$.

Solution:

- (i) $y_1 \leq x$. Violation 0.5
 (ii) $y \leq 9 - 1(3 - x)$
 (iii) $y_1 + y_2 \leq 25 - 1(4 - x)$

- (iv) Gomory fractional cut $\lceil \frac{9}{12} \rceil x_1 + \lceil \frac{12}{12} \rceil x_2 + \lceil \frac{8}{12} \rceil x_3 + \lceil \frac{17}{12} \rceil x_4 + \lceil \frac{13}{12} \rceil x_5 \geq \lceil \frac{50}{12} \rceil = 4.$
(v) Gomory fractional cut $0x_1 + x_2 + x_3 + 0x_4 \leq \lfloor \frac{33}{7} \rfloor = 4.$

4. Consider the problem

$$\begin{aligned} \min & x_1 + 2x_2 \\ x_1 + x_2 & \geq 4 \\ x_1 + 5x_2 & \geq 5 \\ x & \in \mathbb{Z}_+^2. \end{aligned}$$

- (i) Show that $x^* = (\frac{15}{4}, \frac{1}{4})$ is the optimal linear programming solution.
(ii) Generate Gomory mixed integer cuts from the 2 rows of the optimal LP tableau taking the slack variables to be continuous variables.
(iii) Observing that the slack variables are integer, generate the Gomory fractional cuts and compare.
(iv) Solve the instance.

Solution: Optimal LP tableau:

$$\begin{array}{ccccccc} \min & \frac{17}{4} & +\frac{3}{4}x_3 & +\frac{1}{4}x_4 & & & \\ & x_1 & -\frac{5}{4}x_3 & +\frac{1}{4}x_4 & = & \frac{15}{4} & \\ & x_2 & +\frac{1}{4}x_3 & -\frac{1}{4}x_4 & = & \frac{1}{4} & \\ & x_1, x_2 & \in \mathbb{Z}_+^1, x_3, x_4 & \in \mathbb{R}_+^1 & & & \end{array}$$

Gomory mixed integer cuts

$$\text{row 1: } \frac{15}{4}x_3 + \frac{1}{4}x_4 \geq \frac{3}{4}$$

$$\text{row 2: } \frac{1}{4}x_3 + \frac{3}{4}x_4 \geq \frac{1}{4}$$

Gomory fractional (all-integer) cuts

$$\text{row 1: } \frac{3}{4}x_3 + \frac{1}{4}x_4 \geq \frac{3}{4}$$

$$\text{row 2: } \frac{1}{4}x_3 + \frac{3}{4}x_4 \geq \frac{1}{4}$$

Adding the Gomory fractional cut from row 1 and using the dual simplex algorithm gives an optimal solution $x_1 = 3, x_2 = 1, x_3 = 0, x_4 = 3$.

6. Solve $\max\{5x_1 + 9x_2 + 23x_3 - 4y : 2x_1 + 3x_2 + 9x_3 \leq 32 + y, x \in \mathbb{Z}_+^3, y \in \mathbb{R}_+^1\}$ using MIR inequalities.

Solution: LP solution $x = (0, \frac{32}{3}, 0), y = 0$.

$$x_2 + 3x_3 \leq \left\lfloor \frac{32}{3} \right\rfloor + \frac{y}{3 \left(1 - \frac{2}{3}\right)} = 10 + 1y.$$

Then $x = (1, 10, 0), y = 0$

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. Let $x_j = 1$ if j lies in the stable set. 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.

Solution: Let h (odd) be the number of nodes.

One has the inequalities $x_1 + x_2 \leq 1, x_2 + x_3 \leq 1, \dots, x_h + x_1 \leq 1$.

Summing and dividing by 2 gives the valid inequality $x_1 + x_2 + \dots + x_h \leq \frac{h}{2}$.

Chvatal–Gomory rounding gives $x_1 + x_2 + \dots + x_h \leq \lfloor \frac{h}{2} \rfloor = \frac{|H|-1}{2}$.

12. Prove Proposition 8.10.

Solution: Let $Q = \{x : x = y^1 + y^2, A^i y^i \leq b^i z^i, y^i \leq u z_i \text{ for } i = 1, 2, z^1 + z^2 = 1, z^1, z^2 \geq 0\}$. Suppose $(x, y, z) \in Q$ with $0 < z^i < 1$. Set $x^i = \frac{y^i}{z^i}$. Then $x = z^1 x^1 + z^2 x^2$ with $z^1 + z^2 = 1, z^1, z^2 \geq 0$ with $x^i \in P^i$ as $A^i x^i \leq b^i$ and $x^i \leq u$ for $i = 1, 2$. So $x \in \text{conv}(P_1 \cup P_2)$.

When $z^1 = 1$ or 0, it follows directly that $x \in P_1, P_2$, respectively.

The other direction is simple.



17. An Extended Formulation.

Given the 0–1 integer program with $X = \{x \in \{0, 1\}^n : Ax \leq b\}$. For simplicity take $m = 1$.

- (i) Choose some variable x_k and show that the nonlinear inequalities

$$\sum_{j=1}^n a_j x_j x_k \leq b x_k \text{ and } \sum_{j=1}^n a_j x_j (1 - x_k) \leq b (1 - x_k)$$

are valid for X .

- (ii) Show that $x_k^2 = x_k$ is valid for X .

- (iii) Introduce additional variables $y_{ik} = x_i x_k$ for all $i \neq k$. Show that Q^k is a valid relaxation for X :

$$\begin{aligned} \sum_{j \neq k} a_j y_{jk} + a_k x_k &\leq b x_k \\ \sum_j a_j - \sum_{j \neq k} a_j y_{jk} - a_k x_k &\leq b (1 - x_k) \\ y_{ik} &\leq x_k \quad \text{for } i \neq k \\ y_{ik} &\leq x_i \quad \text{for } i \neq k \\ y_{ik} &\geq x_i + x_k - 1 \quad \text{for } i \neq k \\ x \in [0, 1]^n, y_{ik} &\in \mathbb{R}_+^{n-1}. \end{aligned}$$





- (iv) Observe a certain resemblance to the formulation in Proposition 8.10.
- (v) Repeat the reformulation for all variables x_j and set $y_{ij} = y_{ji}$ for all $i \neq j$.
- (vi) Repeat with pairs of variables, multiplying by $x_j x_k, x_j(1 - x_k), (1 - x_j)x_k$, and $(1 - x_j)(1 - x_k)$ and introducing variables y_{ijk} , etc.

Solution:

- (i) As $\sum_j a_j x_j \leq b$ is a valid inequality and $x_k \geq 0$, $\sum_j a_j x_j x_k \leq b x_k$ is a VI.
Similarly as $(1 - x_k) \geq 0$ $\sum_j a_j x_j (1 - x_k) \leq b(1 - x_k)$ is a VI.
- (ii) $x_k^2 = x_k$ for $x_k \in \{0, 1\}$.
- (iii) If $y_{ik} = x_i x_k$ with $x_i, x_k \in \{0, 1\}$, then $y_{ik} \leq x_i$, $y_{ik} \leq x_k$ and $y_{ik} \geq x_i + x_k - 1, y_{ik} \geq 0$ are VIs.
Conversely, if $y_{ik} \leq x_i$, $y_{ik} \leq x_k$ and $y_{ik} \geq x_i + x_k - 1, y_{ik} \geq 0$ one can check:
If $x = (0, 0)$, then $0 \leq y_{ik} \leq x_i = 0$, so $y_{ik} = 0$. Correct.
If $x = (1, 1)$, then $1 = x_i \geq y_{ik} \geq x_i + x_k - 1 = 1$, so $y_{ik} = 1$. Correct.
The two other cases are similar.

Solutions to Certain Exercises in Chapter 9



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) $X = \{x \in \{0, 1\}^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$

$$x = \left(0, \frac{5}{8}, \frac{3}{4}, \frac{3}{4}, 0\right),$$

(ii) $X = \{x \in \{0, 1\}^5 : 9x_1 + 8x_2 + 6x_3 + 6x_4 + 5x_5 \leq 14\}$

$$x = \left(\frac{1}{4}, \frac{1}{8}, \frac{3}{4}, \frac{3}{4}, 0\right),$$

(iii) $X = \{x \in \{0, 1\}^5 : 7x_1 + 6x_2 + 6x_3 + 4x_4 + 3x_5 \leq 14\}$

$$x = \left(\frac{1}{7}, 1, \frac{1}{2}, \frac{1}{4}, 1\right),$$

(iv) $X = \{x \in \{0, 1\}^5 : 12x_1 - 9x_2 + 8x_3 + 6x_4 - 3x_5 \leq 2\}$

$$x = \left(0, 0, \frac{1}{2}, \frac{1}{6}, 1\right).$$

Solution:

- (i) $x_2 + x_3 + x_4 \leq 2$.
- (ii) $2x_1 + 2x_2 + x_3 + x_4 \leq 2$.



(iii) $(x_1) + x_2 + x_3 + x_5 \leq 2$

(iv) Complementing gives $12z_1 + 9z_2 + 8z_3 + 6z_4 + 3z_5 \leq 14$ with

$$z^* = (0, 1, \frac{1}{2}, \frac{1}{6}, 0).$$

$z_2 + z_3 \leq 1$ is violated. In terms of the x -variables, $x_3 \leq x_2$.

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) $X = \{(x, y) \in \{0, 1\}^3 \times \mathbb{R}_+^3 : y_1 + y_2 + y_3 \leq 7, y_1 \leq 3x_1, y_2 \leq 5x_2,$

$$y_3 \leq 6x_3\}$$

$$(x^*, y^*) = (\frac{2}{3}, 1, 0; 2, 5, 0).$$

(ii) $X = \{(x, y) \in \{0, 1\}^3 \times \mathbb{R}_+^3 : 7 \leq y_1 + y_2 + y_3, y_1 \leq 3x_1, y_2 \leq 5x_2,$

$$y_3 \leq 6x_3\}$$

$$(x^*, y^*) = (\frac{2}{3}, 1, 0; 2, 5, 0).$$

(iii) $X = \{(x, y) \in \{0, 1\}^6 \times \mathbb{R}_+^6 : y_1 + y_2 + y_3 \leq 4 + y_4 + y_5 + y_6,$

$$y_1 \leq 3x_1, y_2 \leq 3x_2, y_3 \leq 6x_3, y_4 \leq 3x_4, y_5 \leq 5x_5, y_6 \leq 1x_6.$$

$$(x^*, y^*) = (1, 1, 0, 0, \frac{2}{5}, 0; 3, 3, 0, 0, 2, 0).$$

Solution:

(i) $y_1 + y_2 + 2(1 - x_1) + 4(1 - x_2) \leq 7$ is violated by $\frac{2}{3}$.

(ii) Aggregating constraints gives $3x_1 + 5x_2 + 6x_3 \geq 7, x \in \{0, 1\}^3$.

Valid inequality $x_1 + x_2 + x_3 \geq 2$ is violated by $\frac{1}{3}$.

(iii) $y_1 + y_2 + (1 - x_1) + (1 - x_2) \leq 4 + y_4 + 2x_5 + y_6$ is violated by $\frac{6}{5}$.

16. Consider GAP with equality constraints

$$\begin{aligned} \max & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} = 1 \text{ for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \text{ for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Solve an instance with $m = 3, n = 2, (a_{ij}) = \begin{pmatrix} 5 & 7 \\ 3 & 8 \\ 2 & 10 \end{pmatrix}, (c_{ij}) = \begin{pmatrix} 20 & 16 \\ 15 & 19 \\ 19 & 14 \end{pmatrix}$,

and $b = \begin{pmatrix} 6 \\ 21 \end{pmatrix}$ by cutting planes.

Solution:*Iteration 1*

LP solution: $Z = 57.2$, $x_{11} = 0.8$, $x_{12} = 0.2$, $x_{21} = 0$, $x_{22} = 1$, $x_{31} = 1$, $x_{32} = 0$.
 SP¹: VI for the first knapsack set $\{x \in \{0, 1\}^3 : 5x_1 + 3x_2 + 2x_3 \leq 6\}$ cutting off $x^* = (0.8, 0, 1)$.

Cover inequality $x_{11} + x_{31} \leq 1$ is violated by 0.8. Cut added to LP.

Iteration 2

LP solution: $Z = 54$, $x_{12} = x_{22} = x_{31} = 1$. Integer and optimal.

17. (*Superadditive Inequalities*) See Definition 2.6. Consider the feasible region $X = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$ and let $\mathcal{F} = \{F : \mathbb{R}^m \rightarrow \mathbb{R}^1 : F \text{ is superadditive, } F \text{ is nondecreasing, } F(0) = 0\}$.

- (i) Show that $\sum_{j=1}^n F(a_j)x_j \leq F(b)$ is a valid inequality for X .
- (ii) Show that $F : \mathbb{R}^m \rightarrow \mathbb{R}^1$ with $F(d) = \lfloor \sum_{i=1}^n u_i d_i \rfloor$ is superadditive for $u \in \mathbb{R}_+^m$.
- (iii) Show that if $F^1, F^2 \in \mathcal{F}$, then $F^* = \min(F^1, F^2) \in \mathcal{F}$.

Solution:

- (i) For $x \in X$, $\sum_{j=1}^n F(a_j)x_j = \sum_{j=1: x_j > 0}^n (F(a_j + \dots + F(a_j))) \leq \sum_{j=1}^n F(a_j x_j) \leq F\left(\sum_{j=1}^n a_j x_j\right) \leq F(b)$ where the first inequality uses $x \in \mathbb{Z}_+^n$ and superadditivity, the second uses superadditivity and $F(0) = 0$ and the third uses F nondecreasing and $Ax \leq b$.
- (ii) $F(d^1 + d^2) = \lfloor \sum_i u_i d_i^1 + \sum_i u_i d_i^2 \rfloor = \lfloor \sum_i u_i d_i^1 \rfloor + \lfloor \sum_i u_i d_i^2 \rfloor$
 $(+1 \text{ if } \text{frac}(\sum_i u_i d_i^1 + \sum_i u_i d_i^2) \geq 1) \geq \lfloor \sum_i u_i d_i^1 \rfloor + \lfloor \sum_i u_i d_i^2 \rfloor = F(d^1) + F(d^2)$.
- (iii) Wlog $\min[F_1(u + v), F_2(u + v)] = F_1(u + v) \geq F_1(u) + F_1(v) \geq \min[F_1(u), F_2(u)] + \min[F_1(v), F_2(v)]$.

Solutions to Certain Exercises in Chapter 10

2. Suppose one dualizes the constraints $y_{ij} \leq x_j$ in the strong formulation of *UFL*.
- (i) How strong is the resulting Lagrangian dual bound.
 - (ii) How easy is the solution of the Lagrangian subproblem?

Solution:

- (i) $W_{LD} = Z_{LP}$ as $\sum_j y_{ij} = 1, y_{ij} \in \mathbb{R}_+^n$ is an integer polytope, LP has integer solutions.

(ii) $L(u) = \sum_i L_i(u) + \min \left\{ \sum_j x_j \left(f_j - \sum_i u_{ij} \right) : x \in \{0, 1\}^n \right\}$ where

$$L_i(u) = \min \left\{ \sum_j (c_{ij} + u_{ij}) y_{ij} : \sum_j y_{ij} = 1, y_{ij} \in \mathbb{R}_+^n \right\}$$

$$\text{So } L(u) = \sum_i \min_j (c_{ij} + u_{ij}) + \sum_j (f_j - \sum_i u_{ij})^-.$$

4. Consider GAP with equality constraints

$$\begin{aligned} & \max \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} = 1 \text{ for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \text{ for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Solve an instance with $m = 3, n = 2, (c_{ij}) = \begin{pmatrix} 20 & 16 \\ 15 & 19 \\ 19 & 14 \end{pmatrix}$

$$(a_{ij}) = \begin{pmatrix} 5 & 7 \\ 3 & 8 \\ 2 & 10 \end{pmatrix},$$

$$\text{and } b = \begin{pmatrix} 6 \\ 21 \end{pmatrix}$$

by Lagrangian relaxation.

Solution: The LP solution is $Z = 57.2$ with dual variables $u = (16, 19, 17.4), v = (0.8, 0)$.

Set $u^1 = (16, 19, 17.4)$. Then, $(c_{ij} - u_i) = (4, -4, 1.6; 0, 0, -3.4)$ and a solution $x(u^1) = (1, 0, 0; 0, 1, 0)$ with $L(u^1) = 56.4$, so $\gamma = (0, 0, 1)$.

$u^2 = (16, 19, 17.4) - \mu(0, 0, 1)$. By inspection $L(u)$ decreases for $\mu \leq 2.4$. Take $u^2 = (16, 19, 15)$. Then, $(c_{ij} - u_i) = (4, -4, 4; 0, 0, -1)$ and a solution $x(u^2) = (0, 0, 1; 1, 1, 0)$ with $L(u^2) = 54$ is feasible and thus optimal.

5. Consider a 0-1 knapsack problem

$$Z = \max 11x_1 + 5x_2 + 14x_3$$

$$3x_1 + 2x_2 + 4x_3 \leq 5$$

$$x \in \{0, 1\}^3.$$

Construct a Lagrangian dual by dualizing the knapsack constraint.

- (i) What is the optimal value of the dual variable?
- (ii) Suppose one runs the subgradient algorithm using step size (b) in Theorem 10.4, starting with $u^0 = 2$, $\mu_0 = \frac{1}{2}$ and $\rho = \frac{1}{2}$. Show numerically that the subgradient algorithm does not reach the optimal dual solution.
- (iii) Given that $L(u)$ is a 1-dimensional piecewise-linear convex function, suggest an alternative to the subgradient algorithm.

Solution:

- (i) As $\{x : x \in [0, 1]^3\} = \text{conv}\{x : x \in \{0, 1\}^3\}$, $W_{LD} = Z_{LP}$. The fractional variable in the LP solution is x_3 and so the optimal dual variable is $u = \frac{14}{4}$.
- (ii) Iteration 1. $u^1 = 2$. $L(u^1) = \max\{(11 - 2 \times 3)x_1 + (5 - 2 \times 2)x_2 + (14 - 2 \times 4)x_3 + 2 \times 5 : x \in \{0, 1\}^3\} = 22$.

$$\text{Iteration 2. } u^2 = \max[2 - \frac{1}{2}(5 - 3 - 2 - 4), 0] = 4, L(u^2) = 20$$

$$\text{Iteration 3. } u^3 = \frac{11}{4}, L(u^3) = \frac{39}{2}$$

$$\text{Iteration 4. } u^4 = 3, L(u^4) = 19$$

$$\text{Iteration 5. } u^5 = \frac{25}{8}, L(u^5) = \frac{75}{4}$$

$$\text{Iteration 6. } u^6 = \frac{51}{16}, \text{etc.}$$

- (iii) Try bisection.

6. *Lagrangian Decomposition.* Consider the problem $Z = \max\{cx : A^i x \leq b^i \text{ for } i = 1, 2, x \in \mathbb{Z}_+^n\}$ with the reformulation

$$\begin{aligned} & \max\{\alpha cx^1 + (1 - \alpha)cx^2 : A^i x^i \leq b^i \text{ for } i = 1, 2, \\ & \quad x^1 - x^2 = 0, x^i \in \mathbb{Z}_+^n \text{ for } i = 1, 2\} \end{aligned}$$

for $0 < \alpha < 1$.

Consider the Lagrangian dual of this formulation in which the n constraints $x^1 - x^2 = 0$ are dualized. What is the strength of this dual?

Solution: $L(u) = L^1(u) + L^2(u)$ where

$$L^1(u) = \max\{(\alpha c - u)x^1 : A^1 x^1 \leq b^1, x^1 \in \mathbb{Z}_+^n\} \text{ and}$$

$$L^2(u) = \max\{((1 - \alpha)c + u)x^2 : A^2 x^2 \leq b^2, x^2 \in \mathbb{Z}_+^n\}.$$

$$W_{LD} = \max_{u \in \mathbb{R}^n} L(u).$$

By Theorem 10.3,

$$\begin{aligned} W_{LD} &= \max\{\alpha cx^1 + (1 - \alpha)cx^2 : x^1 \in \text{conv}(A^1x^1 \leq b^1, x^1 \in \mathbb{Z}_+^n), \\ &\quad x^2 \in \text{conv}(A^2x^2 \leq b^2, x^2 \in \mathbb{Z}_+^n), x^1 = x^2\} \\ &= \max\{cx : x \in \text{conv}(A^1x \leq b^1, x \in \mathbb{Z}_+^n) \cap \text{conv}(A^2x \leq b^2, x \in \mathbb{Z}_+^n)\}. \end{aligned}$$

Solutions to Certain Exercises in Chapter 11

1. Consider the problem *UFL* with formulation as in Chapter 1

$$\begin{aligned} \min \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij}y_{ij} + \sum_{j=1}^n f_jx_j \\ \text{subject to} \quad & \sum_{j=1}^n x_{ij} = 1 \text{ for } j = 1, \dots, n \\ & y_{ij} - x_j \leq 0 \text{ for } i = 1, \dots, n, j = 1, \dots, n \\ & y \in \mathbb{R}_+^{mn}, x \in \{0, 1\}^n. \end{aligned}$$



Consider column generation, taking the constraints $\sum_{j=1}^n x_{ij} = 1$ as the complicating constraints.



- (i) Describe the extreme points of $\text{conv}(X)$ where $X = \{(x, y) \in \{0, 1\} \times \mathbb{R}_+^n : y_i \leq x \text{ for } i = 1, \dots, m\}$.
- (ii) What is the complete Master Problem?
- (iii) What is the column generation subproblem?
- (iv) Consider an instance with $m = 4, n = 3$,

$$(c_{ij}) = \begin{pmatrix} 2 & 1 & 5 \\ 3 & 4 & 2 \\ 6 & 4 & 1 \\ 1 & 3 & 7 \end{pmatrix} \quad \text{and} \quad f = (8, 6, 5).$$

Construct the restricted Master Problem using all the initial columns in which a depot serves exactly two clients. Carry out an iteration of the algorithm.

Solution:

- (i) The extreme points are $(y, x) = (e^S, 1)$ for all subsets $S \subseteq M = \{1, \dots, m\}$ and $(\mathbf{0}, 0)$.



(ii) The Master Problem is

$$\begin{aligned} & \sum_j \sum_{S \subseteq M} (\sum_{i \in S} c_{ij} + f_j) \lambda_{j,S} \\ & \sum_j \sum_{S: i \in S} \lambda_{j,S} = 1 \quad \text{for } i \in M \\ & \sum_{S \subseteq M} \lambda_{j,S} \leq 1 \quad \text{for } j = 1, \dots, n \\ & \lambda_{j,S} \geq 0 \quad \text{for } S \subseteq M, j = 1, \dots, n \end{aligned}$$

(iii) The subproblems are $\min \left\{ \sum_{i=1}^m (c_{ij} - u_i)y_{ij} + f_j x_j : (y_j, x_j) \in X \right\}$ for $j = 1, \dots, n$.

3. Consider GAP with equality constraints

$$\begin{aligned} & \max \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ & \sum_{j=1}^n x_{ij} = 1 \quad \text{for } i = 1, \dots, m \\ & \sum_{i=1}^m a_{ij} x_{ij} \leq b_j \quad \text{for } j = 1, \dots, n \\ & x \in \{0, 1\}^{mn}. \end{aligned}$$

Solve an instance with $m = 3, n = 2, (a_{ij}) = \begin{pmatrix} 5 & 7 \\ 3 & 8 \\ 2 & 10 \end{pmatrix}, (c_{ij}) = \begin{pmatrix} 20 & 16 \\ 15 & 19 \\ 19 & 14 \end{pmatrix}$,

and $b = \begin{pmatrix} 6 \\ 21 \end{pmatrix}$ by integer programming decomposition.

Solution: The LP Master problem is initialized with the solutions $(1,0,0)$ and $(0,1,1)$ for the first knapsack constraint and $(0,1,1)$ and $(1,0,0)$ for the second. This gives

$$\begin{array}{rccccccccc} & \max & 20z_1^1 + 34z_2^1 + 33z_1^2 + 16z_2^2 \\ & & 1 & 0 & 0 & 1 & = & 1 \\ & & 0 & 1 & 1 & 0 & = & 1 \\ & & 0 & 1 & 1 & 0 & = & 1 \\ & & 1 & 1 & & & \leq & 1 \\ & & & & 1 & 1 & \leq & 1 \\ & z & \in & & & & & \mathbb{R}_+^4 \end{array}$$

Iteration 1

LP solution has $Z = 53$, $u = (19, 0, 33)$, $v = (1, 0)$

SP¹ takes the form:

$$\zeta^1 = \max[(20, 15, 19) - (19, 0, 33)]x - 1$$

$$5x_1 + 3x_2 + 2x_3 \leq 6$$

$$x \in \{0, 1\}^3$$

SP¹. Solution $\zeta^1 = 14$, $x^* = (0, 1, 0)$. Column added to LPM. z_3^1

Iteration 2

RLM: $Z = 53$, $u = (5, 0, 19)$, $v = (15, 14)$

SP¹. $\zeta^1 = 0$, SP² $\zeta^2 = 16$, $x^* = (1, 1, 0)$. Column added to LPM. z_3^2

Iteration 3

RLM: $Z = 53$, $u = (0, 16, -2)$, $v = (20, 19)$

SP¹: $\zeta^1 = 2$, $x^* = (0, 0, 1)$. Column added to LPM. z_4^1

Iteration 4

RLM: $Z = 54$, $u = (0, 15, -1)$, $v = (20, 20)$

SP¹: $\zeta^1 = 0$

SP²: $\zeta^2 = 0$

RLM is solved to optimality. $z_4^1 = z_3^2 = 1$ gives $x_{31} = x_{12} = x_{22} = 1$.

Solution is integer, so M is solved.



Solutions to Certain Exercises in Chapter 12

1. Write an extended formulation with extreme points and extreme rays for the polyhedron

$$-x_1 + x_2 \leq 1$$

$$3x_1 + x_2 \geq 5$$

$$x_1 + x_2 \geq 1$$

$$x_1 + 2x_2 \leq 11.$$

Solution: P has extreme points $(3, 4)$, $(1, 2)$, and $(2, -1)$ and extreme rays $(4, -3)$ and $(2, -1)$ giving the extended formulation:

$$P = \{x \in \mathbb{R}^2 : x = \begin{pmatrix} 3 \\ 4 \end{pmatrix} \lambda_1 + \begin{pmatrix} 2 \\ -1 \end{pmatrix} \lambda_2 + \begin{pmatrix} 1 \\ 2 \end{pmatrix} \lambda_3 + \begin{pmatrix} 4 \\ -3 \end{pmatrix} \mu_1 + \begin{pmatrix} 2 \\ -1 \end{pmatrix} \mu_2, \\ \lambda_1 + \lambda_2 + \lambda_3 = 1, (\lambda, \mu) \in \mathbb{R}_+^3 \times \mathbb{R}_+^2\}.$$

2. Consider the mixed integer program

$$\begin{array}{lllll} \max & 4x_1 & +5x_2 & +2y_1 & -7y_2 & +5y_3 \\ & 3x_1 & +4x_2 & +2y_1 & -2y_2 & +3y_3 \leq 10 \\ & x \leq 3, x \in \mathbb{Z}_+^2, y \leq 2, y \in \mathbb{R}_+^3. \end{array}$$

Solve it using Benders' algorithm.



Solution:**Top node**

BR $z^* = 127, x^* = (3, 3), \eta^* = 100.$

SP Infeasible. $u^* = 1, uy^* = (0, 2, 0)$ (dual variables on $y \leq 2$).

Feasibility cut $3x_1 + 4x_2 \leq 14$ added.

BR $z^* = 118.25, x^* = (3, 1.25), \eta^* = 100.$

SP $\phi^* = -14, u^* = 3.5, uy^* = (0, 0, 0).$

Optimality cut $\eta + 10.5x_1 + 14x_2 \leq 35$ added.

BR $z^* = 35, x^* = (0, 0), \eta^* = 3.5.$

SP $\phi^* = 14, u^* = 0, uy^* = (2, 0, 5)$

Optimality cut $\eta \leq 14$ added.

BR $z^* = 22, x^* = (2, 0), \eta^* = 14.$

SP $\phi^* = \frac{20}{3}, u^* = \frac{5}{3}, uy^* = (0, 0, 0)$

Optimality cut $\eta + 5x_1 + \frac{20}{3}x_2 \leq \frac{50}{3}$ added.

BR $z^* = 16.1333, x^* = (0.5333, 0), \eta^* = 14.$

SP $\phi^* = 12.4, u^* = 1, uy^* = (0, 0, 2)$

Optimality cut $\eta + 3x_1 + 4x_2 \leq 14$ added.

BR $z^* = \frac{46}{3}, x^* = (\frac{4}{3}, 0), \eta^* = 10.$

SP $\phi^* = 10.$ Top node LP is solved.

$x^* \notin \mathbb{Z}_+^2.$ Branch on $x_1.$

Node $x_1 \leq 1$

BR $z^* = 15.25, x^* = (1, 0.25), \eta^* = 10.$

SP $\phi^* = 10.$ LP is solved.

$x^* \notin \mathbb{Z}_+^2.$ Branch on $x_2.$

Node $x_1 \leq 1, x_2 \leq 0$

BR $z^* = 15, x^* = (1, 0), \eta^* = 11.$

SP $\phi^* = 11, y^* = (0.5, 0, 2)$ LP is solved. New incumbent. $\underline{z}^* = 15.$

Node is pruned by optimality.

Node $x_1 \leq 1, x_2 \geq 1$

BR $z^* = 15,$ Node pruned by bound.

Node $x_1 \geq 2$

BR $z^* = \frac{44}{3},$ Node pruned by bound.

The enumeration is complete.

Optimal solution $x^* = (1, 0), y^* = (0.5, 0, 2), Z = 15.$

4. Consider the facet-generating separation LP (12.9)–(12.14) and suppose that $x^0 = 0$ and $x^* \notin P.$ Write the dual of this LP. Show that it can be interpreted as
 - (i) finding the smallest value $\lambda > 1$ such that $x^* \in \lambda P,$ or as
 - (ii) finding the last point on the line from the origin to x^* that lies in $P.$

Solution:

(i) The dual of

$$\max \gamma x^* - \gamma_0$$

$$\gamma - uF \leq 0$$

$$-uG \leq 0$$

$$u(d - Fx^0) - \gamma_0 \leq 0$$

$$\gamma_0 \leq 1$$

$$u \in \mathbb{R}_+^m.$$

is, with dual variables (x, y, λ, μ) ,

$$\min \mu$$

$$x = x^*$$

$$-\lambda + \mu = -1$$

$$-Fx - Gy + \mu d \geq 0$$

$$x, y, \mu, \lambda \geq 0$$

which can be rewritten as

$$\min \lambda - 1$$

$$Fx^* + Gy \leq \lambda d$$

$$y \geq 0, \lambda \geq 1.$$

Let $\lambda^* > 1$ be an optimal solution.

- (ii) Note that if $\hat{x} \in P$ is the point such that $\lambda^* \hat{x} = x^*$, then \hat{x} is the last point in P on the line from the origin to x^* .

Solutions to Certain Exercises in Chapter 13

1. Devise a simple heuristic for the instance of GAP in Exercise 4 in Chapter 10.

Solution: A very naive approach.*Find a feasible solution.*

Take the smallest a_{ij} in each column and set the corresponding $x_{ij} = 1$. If the solution is infeasible and $x_{ij} = 1$, try to swap for another variable in the row $x_{i,k}$ and test for feasibility.



Improving a feasible solution.

Again if $x_{ij} = 1$, try to swap for another variable in the row x_{ik} that maintains feasibility and increases the value $c_{ik} > c_{ij}$.

For this instance, initial solution $x_{11} = x_{21} = x_{31} = 1$. Infeasible in column 1. Swap x_{11} and set $x_{12} = 1$. Solution feasible.

Feasible solution $x_{12} = x_{21} = x_{31} = 1$. Swap x_{21} and set $x_{22} = 1$ as solution remains feasible and $c_{22} = 19 > c_{21} = 15$.

Feasible solution $x_{12} = x_{22} = x_{31} = 1$. Swapping x_{12} makes the solution infeasible and swapping x_{31} decreases the value. Stop. Heuristic solution $x_{12} = x_{22} = x_{31} = 1$ with objective value 19.

Solutions to Certain Exercises in Chapter 14

1. As part of a multicommodity network design problem, you encounter the subset of different commodities passing through an arc. Each of the flows has an upper bound d_j corresponding to its total demand. To meet the aggregate flow on the arc, capacity in multiples of C units needs to be installed. Thus, the set can be formulated as follows: $X = \{(x, y) \in \mathbb{Z} \times \mathbb{R}_+^n : \sum_{j=1}^n y_j \leq Cx, y_j \leq d_j \text{ for } j = 1, \dots, n\}$. For an instance with $n = 6$, $C = 10$, $d = (4, 9, 6, 3, 5, 7)$, you obtain the fractional solution $y^* = (0, 9, 6, 0, 0, 7)$, $x^* = 2.2$. Find a valid inequality cutting off this point. Describe a family of valid inequalities for the set X .



Solution: As $y_1 = y_4 = y_5 = 0$, we relax the inequality to obtain $w = y_2 + y_3 + y_6 \leq 6x$ and also $w \leq d_2 + d_3 + d_6 = 22$. Now, see Example 8.3, we obtain the inequality $w \leq 22 - 2(3 - x)$ cutting off the point.

More generally, $w = \sum_{j \in S} w_j \leq Cx$ and $w \leq \sum_{j \in S} d_j$ leads to an exponential family of inequalities.

3. Consider an item as in the multi-item lot-sizing problem in Section 14.4.
 - (i) Suppose that the item cannot be produced for more than γ consecutive periods. Formulate. If the formulation appears to be weak, look for a stronger formulation.
 - (ii) If the item must be produced at least once every δ periods, give a strong formulation.

Solution:

- (i) One possibility: If there is a switch-on in period t , the item must not be produced in one of the periods in the interval $[t + 1, t + \gamma]$. This gives:

$$z_t \leq \sum_{u=t+1}^{t+\gamma} (1 - y_u).$$





A better alternative: If the item is produced in period t , production of the item must have started at some period in the interval $[t - \gamma + 1, t]$. This gives

$$y_t \leq \sum_{u=t-\gamma+1}^t z_u.$$

- (ii) If not produced in t , production must start during the interval $[t + 1, t + \delta]$. This gives

$$1 - y_t \leq \sum_{u=t+1}^{t+\delta} z_u.$$

5. You have a discrete time model in which $x_t^i = 1$ if job i starts in period t . All three jobs must be carried out. Given two jobs, you wish to formulate the constraint that the start of job 1 is not more than three periods after the start of job 2 and the start of job 3 is at least five periods after the start of job 1.
- (i) Formulate without introducing additional variables.
 - (ii) Let $z_t^i = 1$ if job i starts in or before period t . Reformulate using the z_t^i variables. Can you say anything about the structure of the constraint matrix?

Solution:

- (i) First, there are the assignment constraints $\sum_t x_t^i = 1$ for $i = 1, 2, 3$. Then, if job 2 has started in or before time t , then job 1 must start in or before $t + 3$. This gives

$$\sum_{u=1}^t x_u^2 \leq \sum_{u=1}^{t+3} x_u^1.$$

Similarly, if job 3 starts in or before period t , then job 1 starts in or before $t - 5$ giving:

$$\sum_{u=1}^t x_u^3 \leq \sum_{u=1}^{t-5} x_u^1.$$

- (ii) $x_t^i \in \{0, 1\}$ becomes $0 \leq z_t^1 \leq z_t^2 \leq \dots \leq z_t^i = 1$. The other two constraints become $z_t^2 \leq z_{t+3}^1$ and $z_t^3 \leq z_{t-5}^1$ for all appropriate t . All the constraints have the form $w^1 - w^2 \leq b$ with b integer. The matrix is the dual of a network matrix and therefore totally unimodular.

