

---

# LARGE SCALE LINEAR AND INTEGER OPTIMIZATION: A UNIFIED APPROACH

---

---

# LARGE SCALE LINEAR AND INTEGER OPTIMIZATION: A UNIFIED APPROACH

---

**Richard Kipp Martin**  
*Graduate School of Business*  
*University of Chicago*



*Springer Science+Business Media, LLC*

**Library of Congress Cataloging-in-Publication Data**

Martin, Richard Kipp.

Large scale linear and integer optimization : a unified approach /  
Richard Kipp Martin.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-4613-7258-5 ISBN 978-1-4615-4975-8 (eBook)  
DOI 10.1007/978-1-4615-4975-8

1. Linear programming. 2. Mathematical optimization. 1. Title.

T57.75.M375 1999

519.7'2--dc21

98-46062

CIP

---

Copyright © 1999 Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers in 1999

Softcover reprint of the hardcover 1st edition 1999

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, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher, Springer Science+Business Media, LLC .

*Printed on acid-free paper.*

This book is dedicated to my parents, Bruce and Phyllis Martin.

---

# CONTENTS

Preface	xv
<b>Part I MOTIVATION</b>	1
<b>1 LINEAR AND INTEGER LINEAR OPTIMIZATION</b>	3
1.1 Introduction	3
1.2 Linear and Integer Linear Optimization	5
1.3 A Guided Tour of Applications	7
1.4 Special Structure	21
1.5 Linear and Integer Linear Programming Codes	25
1.6 Other Directions	28
1.7 Exercises	29
<b>Part II THEORY</b>	33
<b>2 LINEAR SYSTEMS AND PROJECTION</b>	35
2.1 Introduction	35
2.2 Projection for Equality Systems: Gaussian Elimination	36
2.3 Projection for Inequality Systems: Fourier-Motzkin Elimination	39
2.4 Applications of Projection	46
2.5 Theorems of the Alternative	49
2.6 Duality Theory	57
2.7 Complementary Slackness	61
2.8 Sensitivity Analysis	65
2.9 Conclusion	75
2.10 Exercises	75

<b>3</b>	<b>LINEAR SYSTEMS AND INVERSE PROJECTION</b>	81
3.1	Introduction	81
3.2	Deleting Constraints by Adding Variables	81
3.3	Dual Relationships	91
3.4	Sensitivity Analysis	93
3.5	Conclusion	99
3.6	Homework Exercises	100
<b>4</b>	<b>INTEGER LINEAR SYSTEMS: PROJECTION AND INVERSE PROJECTION</b>	103
4.1	Introduction	103
4.2	Background Material	105
4.3	Solving A System of Congruence Equations	114
4.4	Integer Linear Equalities	122
4.5	Integer Linear Inequalities: Projection	124
4.6	Integer Linear Inequalities: Inverse Projection	127
4.7	Conclusion	136
4.8	Exercises	137
<b>Part III ALGORITHMS</b>		141
<b>5</b>	<b>THE SIMPLEX ALGORITHM</b>	143
5.1	Introduction	143
5.2	Motivation	143
5.3	Pivoting	147
5.4	Revised Simplex	154
5.5	Product Form of the Inverse	162
5.6	Degeneracy and Cycling	168
5.7	Complexity of the Simplex Algorithm	177
5.8	Conclusion	178
5.9	Exercises	178
<b>6</b>	<b>MORE ON SIMPLEX</b>	183
6.1	Introduction	183
6.2	Sensitivity Analysis	184
6.3	The Dual Simplex Algorithm	191

6.4	Simple Upper Bounds and Special Structure	198
6.5	Finding a Starting Basis	201
6.6	Pivot Column Selection	205
6.7	Other Computational Issues	209
6.8	Conclusion	215
6.9	Exercises	216
<b>7</b>	<b>INTERIOR POINT ALGORITHMS: POLYHEDRAL TRANSFORMATIONS</b>	<b>219</b>
7.1	Introduction	219
7.2	Projective Transformations	225
7.3	Karmarkar's Algorithm	231
7.4	Polynomial Termination	234
7.5	Purification, Standard Form and Sliding Objective	239
7.6	Affine Polyhedral Transformations	243
7.7	Geometry of the Least Squares Problem	254
7.8	Conclusion	258
7.9	Exercises	258
<b>8</b>	<b>INTERIOR POINT ALGORITHMS: BARRIER METHODS</b>	<b>261</b>
8.1	Introduction	261
8.2	Primal Path Following	266
8.3	Dual Path Following	272
8.4	Primal-Dual Path Following	277
8.5	Polynomial Termination of Path Following Algorithms	283
8.6	Relation to Polyhedral Transformation Algorithms	292
8.7	Predictor-Corrector Algorithms	297
8.8	Other Issues	300
8.9	Conclusion	306
8.10	Exercises	310
<b>9</b>	<b>INTEGER PROGRAMMING</b>	<b>313</b>
9.1	Introduction	313
9.2	Modeling with Integer Variables	314
9.3	Branch-and-Bound	319
9.4	Node and Variable Selection	324

9.5	More General Branching	328
9.6	Conclusion	341
9.7	Exercises	341
<b>Part IV SOLVING LARGE SCALE PROBLEMS: DECOMPOSITION METHODS</b>		347
<b>10</b>	<b>PROJECTION: BENDERS' DECOMPOSITION</b>	349
10.1	Introduction	349
10.2	The Benders' Algorithm	350
10.3	A Location Application	354
10.4	Dual Variable Selection	360
10.5	Conclusion	364
10.6	Exercises	365
<b>11</b>	<b>INVERSE PROJECTION: DANTZIG-WOLFE DECOMPOSITION</b>	369
11.1	Introduction	369
11.2	Dantzig-Wolfe Decomposition	370
11.3	A Location Application	375
11.4	Taking Advantage of Block Angular Structure	384
11.5	Computational Issues	386
11.6	Conclusion	390
11.7	Exercises	391
<b>12</b>	<b>LAGRANGIAN METHODS</b>	393
12.1	Introduction	393
12.2	The Lagrangian Dual	394
12.3	Extension to Integer Programming	398
12.4	Properties of the Lagrangian Dual	402
12.5	Optimizing the Lagrangian Dual	408
12.6	Computational Issues	426
12.7	A Decomposition Algorithm for Integer Programming	429
12.8	Conclusion	434
12.9	Exercises	435

<b>Part V SOLVING LARGE SCALE PROBLEMS: USING SPECIAL STRUCTURE</b>	437
<b>13 SPARSE METHODS</b>	439
13.1 Introduction	439
13.2 <i>LU</i> Decomposition	439
13.3 Sparse LU Update	446
13.4 Numeric Cholesky Factorization	462
13.5 Symbolic Cholesky Factorization	466
13.6 Storing Sparse Matrices	471
13.7 Programming Issues	472
13.8 Computational Results: Barrier versus Simplex	478
13.9 Conclusion	479
13.10 Exercises	480
<b>14 NETWORK FLOW LINEAR PROGRAMS</b>	481
14.1 Introduction	481
14.2 Totally Unimodular Linear Programs	482
14.3 Network Simplex Algorithm	493
14.4 Important Network Flow Problems	505
14.5 Almost Network Problems	513
14.6 Integer Polyhedra	515
14.7 Conclusion	523
14.8 Exercises	524
<b>15 LARGE INTEGER PROGRAMS: PREPROCESSING AND CUTTING PLANES</b>	527
15.1 Formulation Principles and Techniques	527
15.2 Preprocessing	533
15.3 Cutting Planes	542
15.4 Branch-and-Cut	555
15.5 Lifting	557
15.6 Lagrangian Cuts	560
15.7 Integer Programming Test Problems	561
15.8 Conclusion	562
15.9 Exercises	563

<b>16</b>	<b>LARGE INTEGER PROGRAMS: PROJECTION AND INVERSE PROJECTION</b>	565
16.1	Introduction	565
16.2	Auxiliary Variable Methods	573
16.3	A Projection Theorem	601
16.4	Branch-and-Price	601
16.5	Projection of Extended Formulations: Benders' Decomposition Revisited	613
16.6	Conclusion	630
16.7	Exercises	630
<b>Part VI APPENDIX</b>		633
<b>A</b>	<b>POLYHEDRAL THEORY</b>	635
A.1	Introduction	635
A.2	Concepts and Definitions	635
A.3	Faces of Polyhedra	640
A.4	Finite Basis Theorems	645
A.5	Inner Products, Subspaces and Orthogonal Subspaces	651
A.6	Exercises	653
<b>B</b>	<b>COMPLEXITY THEORY</b>	657
B.1	Introduction	657
B.2	Solution Sizes	660
B.3	The Turing Machine	661
B.4	Complexity Classes	663
B.5	Satisfiability	667
B.6	$\mathcal{NP}$ -Completeness	669
B.7	Complexity of Gaussian Elimination	670
B.8	Exercises	674
<b>C</b>	<b>BASIC GRAPH THEORY</b>	677
<b>D</b>	<b>SOFTWARE AND TEST PROBLEMS</b>	681
<b>E</b>	<b>NOTATION</b>	683

<b>References</b>	687
<b>AUTHOR INDEX</b>	723
<b>TOPIC INDEX</b>	731

---

## PREFACE

This is a textbook about linear and integer linear optimization. There is a growing need in industries such as airline, trucking, and financial engineering to solve very large linear and integer linear optimization problems. Building these models requires uniquely trained individuals. Not only must they have a thorough understanding of the theory behind mathematical programming, they must have substantial knowledge of how to solve very large models in today's computing environment. The major goal of the book is to develop the theory of linear and integer linear optimization in a unified manner and then demonstrate how to use this theory in a modern computing environment to solve very large real world problems.

After presenting introductory material in Part I, Part II of this book is devoted to the theory of linear and integer linear optimization. This theory is developed using two simple, but unifying ideas: projection and inverse projection. Through projection we take a system of linear inequalities and replace some of the variables with additional linear inequalities. Inverse projection, the dual of this process, involves replacing linear inequalities with additional variables. Fundamental results such as weak and strong duality, theorems of the alternative, complementary slackness, sensitivity analysis, finite basis theorems, etc. are all explained using projection or inverse projection. Indeed, a unique feature of this book is that these fundamental results are developed and explained *before* the simplex and interior point algorithms are presented. We feel that projection and inverse projection are the very essence of these fundamental results and proofs based upon simplex or interior point algorithms are not insightful. The ideas of projection and inverse projection are also extended to integer linear optimization. With the projection-inverse projection approach theoretical results in integer linear optimization become much more analogous to their linear optimization counterparts. Thus, once armed with these two concepts the reader is equipped to understand fundamental theorems in an intuitive way.

In Part III of the book we present the most important algorithms that are used in commercial software for solving real world problems. Even though the algorithms developed in Part III are appropriate for solving large realistic

problems, they will often fail for *very large scale* applications unless advantage is taken of the special structure present in the problem. In Part IV we show how to take advantage of special structure through decomposition. We explain these decomposition algorithms as extensions of the projection and inverse projection concepts developed in Part II. In Part V we show how to take advantage of special structure by modifying and enhancing the algorithms developed in Part III. This section contains a discussion of some of the most current research in linear and integer programming. We also show in Part V how to take different problem formulations and appropriately “modify” them so that the algorithms from Part III are much more efficient. Once again, the projection and inverse projection concepts are used in Part V to present the current research in linear and integer linear optimization in a very unified way.

An ambitious one quarter or semester course in linear optimization would include Chapters 1-3 and Chapters 5-8. This set of chapters gives the student exposure to the most important aspects of linear optimization theory as well as the simplex and barrier algorithms. If two quarters are available, then the instructor has several options. The second quarter can be devoted to a brief introduction to integer linear optimization and then move on to large scale linear optimization. This material is in Chapters 9-14. Or, if the instructor wishes to cover integer linear optimization in more depth, then Chapters 4, 9, 12, and 14-16 provide a solid background. Chapters 4, 9, 12, 14-16 provide a nice two quarter, or intense semester course in discrete optimization.

No prior knowledge of linear or integer linear optimization is assumed, although this text is written for a mathematically mature audience. Our target audience is upper level undergraduate students and graduate students in computer science, applied mathematics, industrial engineering and operations research/management science. Coursework in linear algebra and analysis is sufficient background. Researchers wishing to brush up on recent developments in large scale linear and integer linear programming will also find this text useful. Readers not familiar with any polyhedral theory should read the first two sections of Appendix A. The necessary background on complexity theory is provided in Appendix B. A brief tutorial on graph theory is provided in Appendix C.

This book has benefited from the help of numerous people. Former students Mohan Bala, John Borse, Peter Cacioppi, Kevin Cunningham, Zeger Degraeve, Syam Menon, Santosh Nabar, Fritz Raffensperger, Larry Robinson, Jayaram Sankaran, Jack Shi, Sri Sinnathamby, Kamaryn Tanner, Jurgen Tistaert, Mala Viswanath, and Hongsuk Yang suffered through constant revisions, found numerous errors, and provided useful suggestions for improvement. I thank James

Evans, William Lippert, Richard Murphy, James Powell, and the late Richard DeMar for contributing greatly to my mathematical education. I have also benefited from my colleagues Gary Eppen, Ronald Rardin, and the late Robert Jeroslow. Luis Gouveia has been particularly helpful in reading much of the manuscript and making suggestions for improvement. Most of all, I thank Dennis Sweeney for teaching me so much mathematical optimization over the years and Linus Schrage for so graciously enduring a constant barrage of questions related to topics in this book. I thank Gary Folven at Kluwer for being a patient editor and enthusiastically supporting this book. Finally, I thank my parents, Bruce and Phyllis Martin, and Gail Honda for their constant encouragement and support throughout this long project.

## **PART I**

---

### **MOTIVATION**

# LINEAR AND INTEGER LINEAR OPTIMIZATION

## 1.1 INTRODUCTION

This is a textbook about linear and integer linear optimization. The major goal is to develop the theory of linear and integer linear optimization in a unified manner and then demonstrate how to use this theory to solve very large real world problems. No prior knowledge of these topics is assumed, although this text is written for a mathematically mature audience. Our target audience is upper level undergraduate students and graduate students in computer science, applied mathematics, industrial engineering and operations research/management science. Coursework in linear algebra and analysis is sufficient background. Researchers wishing to brush up on recent developments in large scale linear and integer linear optimization will also find this text useful.

In the last decade there has been an incredible explosion in both methodology development and implementation of methodology to solve real problems. Problems that were far too large to solve using a mainframe computer in the 1980s are easily solved today on a desktop computer. Never has it been more important to have an understanding of how to optimize very large linear and integer linear optimization problems with *special structure*. (In Section 1.4 of this chapter we explain precisely the meaning of special structure.) There is a growing need in industries such as airline, trucking, and financial engineering to solve very large linear and integer linear optimization problems. For example, the operations research group at American Airlines has been so successful they have spun off as a separate subsidiary, American Airlines Decision Technologies, Inc. They now provide consulting services to other corporations. A very successful mathematical modeling effort in supply chain management led to pretax annual savings of over 200 million dollars for the Procter & Gamble Co. and motivated the company to establish their Analytics Center of Expertise

(ACOE). See Camm et al. [81]. Building these large models requires uniquely trained individuals. Solving realistic problems requires a thorough understanding of the theory behind mathematical optimization and substantial knowledge of how to solve very large models in today's computing environment.

Part II of this book is devoted to the theory of linear and integer linear optimization. This theory is developed using two simple, but unifying ideas: projection and inverse projection. Through projection we take a system of linear inequalities and replace some of the variables with additional linear inequalities. Inverse projection, the dual of this process, involves replacing linear inequalities with additional variables. Fundamental results such as weak and strong duality, theorems of the alternative, complementary slackness, sensitivity analysis, finite basis theorems, etc. are all explained using projection or inverse projection. Indeed, a unique feature of this book is that these fundamental results are developed and explained *before* the simplex and interior point algorithms are presented. We feel that projection and inverse projection are the very essence of these fundamental results and proofs based upon simplex or interior point algorithms are not insightful. The ideas of projection and inverse projection are also extended to integer linear optimization. With the projection-inverse projection approach theoretical results in integer linear optimization become much more analogous to their linear optimization counterparts. Finally, we further show in Part IV that large scale decomposition algorithms are applications of projection and inverse projection. Thus, once armed with these two concepts the reader is equipped to understand fundamental theorems in an intuitive way.

In Part III of the book we present the most important algorithms that are used in commercial software for solving real world problems. Even though the algorithms developed in Part III are appropriate for solving large realistic problems, they will often fail for *very large scale* applications unless advantage is taken of the special structure present in the problem. In Part IV we show how to take advantage of special structure through decomposition. In Part V we show how to take advantage of special structure by modifying and enhancing the algorithms developed in Part III. We also show in Part V how to take different problem formulations and appropriately "modify" them so that the algorithms from Part III are much more efficient. Once again, the projection and inverse projection concepts are used in Part V to explain some of the recent algorithms for solving very large problems.

In Section 1.2 of this chapter a formal definition of linear and integer linear optimization problems is given. A brief history behind their development is also given. Section 1.3 is a guided tour of several interesting structures and applications of linear and integer linear optimization. The purpose is to provide

the reader with an appreciation of just how applicable this material is, and to introduce important problem structures used throughout the book. In Section 1.4 we explain exactly what is meant by special structure. In Section 1.5 we mention several software packages that are discussed throughout the book and give a sample computer output of a linear optimization solution. In Section 1.6 a guideline is provided to books which concentrate on other aspects of linear and integer linear optimization. Exercises are provided in Section 1.7.

## 1.2 LINEAR AND INTEGER LINEAR OPTIMIZATION

Students of mathematics are required to study linear algebra and are familiar with properties of systems of linear equalities. Rarely are they exposed to an in-depth study of linear inequalities. Although Fourier [159] discovered a method for finding a solution to a system of linear inequalities in 1826, systems of linear inequalities were a topic rarely studied by pure mathematicians. Problems with linear inequalities were first studied in earnest by mathematical economists such as Koopmans, Kantorovich and Leontief. The problem of maximizing or minimizing a linear function subject to linear constraints blossomed in the 1940s when George Dantzig was a Mathematical Advisor to the U.S. Air Force Comptroller and the Noble Prize winning economist George Stigler was studying the problem of the minimal cost of subsistence. In 1945 Stigler published his paper, “Cost of Subsistence” [412], and formulated a *diet problem*, which was constructing a menu for satisfying about a dozen nutritional requirements at minimal cost, as a linear optimization problem. Stigler realized that this was a special case of the more general problem of minimizing a linear function subject to linear inequalities and stated “... there does not appear to be any direct method of finding the minimum of a linear function subject to linear constraints.” At the same time, Dantzig was engaged in efforts to find rapid solutions to military deployment, training and logistical supply problems, which led to his realization that many important planning problems could be formulated as a system of linear inequalities. In 1947 Dantzig formalized the concept of a *linear program* as we know it today when he formulated the planning problem with a linear objective subject to solving a system of linear equations and inequalities. The term *program* does not refer to a computer program as one might suspect (since a computer is required to solve any realistic problem) but is a term used by the military to describe the plan or proposed solution to the deployment and logistical problems. The term linear program “stuck” and is used in the generic sense for any linear optimization problem. The terms linear

programming and linear optimization are used synonymously throughout the book. The linear programming problem (*LP*) is

$$\min c^T x \quad (1.1)$$

$$(LP) \quad \text{s.t. } Ax = b \quad (1.2)$$

$$x \geq 0 \quad (1.3)$$

where  $A$  is an  $m \times n$  matrix of rational numbers,  $c$  is an  $n$  component rational column vector and  $b$  an  $m$  component rational column vector. Equation (1.1) is the *objective function*. In linear program (*LP*) this objective function is minimized subject to, abbreviated by s.t. in (1.2), the *constraint set* (1.2) plus the *nonnegativity constraints* (1.3). The unknowns, or variables in the problem  $x$ , are called *decision variables*. The vector  $b$  in (1.2) is known as the *right hand side*. The canonical structure (*LP*) is a linear program in *standard form*. In many of applications, some of the  $Ax = b$  constraints may actually be inequality constraints. An inequality constraint is  $(a^i)^T x \geq b_i$  is *binding* for a given solution  $\bar{x}$  if  $(a^i)^T \bar{x} = b_i$ . We use the terms *binding* and *tight* synonymously throughout the text. If an inequality constraint does not hold as an equality for a given solution it has positive *slack*. For example, if  $x_1 = 1$  and  $x_2 = 3$  the inequality  $5x_1 + 7x_2 \geq 26$  is binding or tight. If  $x_1 = 2$  and  $x_2 = 3$  the constraint  $5x_1 + 7x_2 \geq 26$  is not binding and has slack of 5.

Not only did Dantzig formalize the concept of a linear program, he also came up with a solution method now known as the simplex algorithm. The first “large scale” problem (nine equations and 77 nonnegative variables) solved was the Stigler diet problem. The problem was solved using desk calculators in about 120 person hours. Today this problem is easily solved in less than a second on a low-end desktop computer. As an aside, the reader interested in a real world implementation of the diet problem linear programming solution should, in his or her next visit to the grocery store, concentrate heavily on dried navy beans, flour, cabbage and spinach. For more on the history of inequalities and linear programming we highly recommend “Reminiscences About the Origins of Linear Programming” by George Dantzig [113] and the thorough historical notes in the text, *Theory of Linear and Integer Programming* by Alexander Schrijver [403]. The 1993 Morris lecture by George Nemhauser published in *Operations Research* [352] describes many of the recent advances in linear and integer linear programming and puts these developments into historical perspective. Finally, the book edited by Lenstra, Rinnooy Kan, and Schrijver [284] contains personal recollections by many of the founders of linear and integer linear programming.

No sooner were linear programs being solved routinely than people realized that in many cases fractional values for the decision variables did not make sense.

Often, rounding fractional decision variables to the nearest integer resulted in little loss of accuracy. However, in many cases where the decision variable represents funding a project or not, opening a warehouse or not, it is important to restrict the variables to be integer and rounding can result in infeasibility or significant profit loss. Much of this book deals with mixed integer linear programming and the *mixed integer linear program* is

$$(MIP) \quad \begin{aligned} & \min c^T x \\ & \text{s.t. } Ax = b \\ & \quad x \geq 0 \\ & \quad x_j \in \mathbb{Z}, \quad j \in I. \end{aligned}$$

In virtually all of the examples we consider, the integer variables are actually binary variables, i.e., they represent a yes-no decision and are required to be 1 (yes) or 0 (no). When all of the variables in  $(MIP)$  are restricted to be integer it is called an *integer linear program*. When all of the integer variables are binary it is a *binary linear program*. If we use just the term “integer program” we loosely take it to mean a linear program with some or all of the variables required to be integer.

In the next section we illustrate the applicability of the material in this book by presenting some important linear and integer programming applications. These examples are in no way exhaustive; they are meant to show just how significant linear programming is and to introduce some of the problem structures that we study. A complete listing of the vast array of applications of linear and integer programming would be impossible.

## 1.3 A GUIDED TOUR OF APPLICATIONS

### 1.3.1 Blending

The diet problem formulated by Stigler was the first linear program solved using the simplex algorithm. The diet problem is a specific example of the more generic class of *blending problems*. As the name implies, a blending problem involves mixing together raw material in such a way that requirements on the output are satisfied. In the case of the diet problem the raw materials are the various foods and the constraints are the nutritional requirements. Blending problems arise frequently in the oil industry where various grades of crude oil are the raw materials which must be blended into gasolines which meet

**Table 1.1** Blending Raw Material Requirements

	Cost per Pound	Percent Carbon	Percent Chrome	Percent Manganese	Percent Silicon	Amount Available	Decision Variable
Pig Iron 1	\$0.030	4	0	0.9	2.25	9000	P1
Pig Iron 2	\$0.065	0	10	4.5	15	9000	P2
Ferro-Silc 1	\$0.065	0	0	0	45	9000	F1
Ferro-Silc 2	\$0.061	0	0	0	42	9000	F2
Alloy 1	\$0.100	0	0	60	18	9000	A1
Alloy 2	\$0.130	0	20	9	30	9000	A2
Alloy 3	\$0.119	0	8	33	25	9000	A3
Carbide	\$0.080	15	0	0	30	20	CB
Steel 1	\$0.021	0.4	0	0.9	0	200	S1
Steel 2	\$0.020	0.1	0	0.3	0	200	S2
Steel 3	\$0.0195	0.1	0	0.3	0	200	S3

**Table 1.2** Blending Quality Requirements

	At Least	Not More Than
Carbon Content	3.0 %	3.5%
Chrome Content	0.3%	0.45%
Manganese Content	1.35%	1.65%
Silicon Content	2.7%	3.0%

octane, lead, etc. requirements. In order to obtain a feel for formulating linear programs and to see an example of a blending problem consider the Pittsburgh Steel example taken from Schrage [402].

The raw materials available to the Pittsburgh Steel Company are listed in the first column of Table 1.1. These raw materials must be blended together in order to make one ton (2000 pounds) of a new grade of steel with the quality requirements given in Table 1.2. A key assumption in formulating the linear program is that the chemical content of the final steel blend is simply the weighted chemical content of its components. For example, consider the requirement that the blend have at least a 3% carbon content. Using the definition of the decision variables given in Table 1.1 (for example  $P_1$  is the pounds of Pig Iron 1 used in the blend), the total weight of the blend is

$$P_1 + P_2 + F_1 + F_2 + A_1 + A_2 + A_3 + CB + S_1 + S_2 + S_3 = 2000$$

and the amount in pounds of carbon in the blend is

$$.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3.$$

Then the percentage by weight of carbon in the blend is

$$(.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3)/2000$$

and the lower bound of at least 3% is expressed by the inequality

$$(.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3)/2000 \geq .03$$

or,

$$.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3 \geq 60.$$

Similarly, the upper limit of 3.5% on the carbon content is expressed as

$$.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3 \leq 70.$$

The complete formulation for this problem is given below.

$$\begin{aligned} \text{min } & 0.03P_1 + 0.0645P_2 + 0.065F_1 + 0.061F_2 + .1A_1 + .13A_2 \\ & + 0.119A_3 + .08CB + 0.021S_1 + 0.02S_2 + 0.0195S_3 \end{aligned} \quad (1.4)$$

s.t.

#### RAW MATERIAL CONSTRAINTS

$$CB \leq 20 \quad (1.5)$$

$$S_1 \leq 200 \quad (1.6)$$

$$S_2 \leq 200 \quad (1.7)$$

$$S_3 \leq 200 \quad (1.8)$$

#### QUALITY REQUIREMENTS

##### CARBON

$$.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3 \geq 60 \quad (1.9)$$

$$.04P_1 + .15CB + 0.004S_1 + .001S_2 + .001S_3 \leq 70 \quad (1.10)$$

##### CHROME CONTENT

$$.1P_2 + .2A_2 + .08A_3 \geq 6 \quad (1.11)$$

$$.1P_2 + .2A_2 + .08A_3 \leq 9 \quad (1.12)$$

##### MANGANESE CONTENT

$$\begin{aligned} 0.009P_1 + 0.045P_2 + 0.6A_1 + 0.09A_2 + 0.33A_3 + 0.009S_1 \\ + 0.003S_2 + 0.003S_3 \geq 27 \end{aligned} \quad (1.13)$$

$$\begin{aligned} 0.009P_1 + 0.045P_2 + 0.6A_1 + 0.09A_2 + 0.33A_3 + 0.009S_1 \\ + 0.003S_2 + 0.003S_3 \leq 33 \end{aligned} \quad (1.14)$$

## SILICON CONTENT

$$0.0225P1 + 0.15P2 + 0.45F1 + 0.42F2 + 0.18A1 + 0.3A2 \\ + 0.25A3 + 0.3CB \geq 54 \quad (1.15)$$

$$0.0225P1 + 0.15P2 + 0.45F1 + 0.42F2 + 0.18A1 + 0.3A2 \\ + 0.25A3 + 0.3CB \leq 60 \quad (1.16)$$

## ONE TON PRODUCTION REQUIREMENT

$$P1 + P2 + F1 + F2 + A1 + A2 + A3 + CB + S1 + S2 + S3 = 2000 \quad (1.17)$$

The objective function (1.4) expresses the goal to make the blend at minimum cost. Constraints (1.5)-(1.8) are the upper limits on the availability of Carbide, Steel 1, Steel 2 and Steel 3. Constraints (1.9)-(1.16) are the upper and lower limit chemical content constraints. Constraint (1.17) requires that 2000 pounds of the blend be made. All variables are required to be nonnegative. The optimal solution is given below. Only the materials which are used in the mix appear in the solution. There is no constraint to use all of the raw materials in the blend.

## OBJECTIVE FUNCTION VALUE

1) 59.55629

VARIABLE	VALUE
P1	1474.264100
P2	60.000000
F2	22.062050
A1	14.238862
S1	200.000000
S2	29.434932
S3	200.000000

Notice that the solution has fractional variables. For example, 29.434932 pounds of steel 2 are used. In blending and many other applications fractional solutions have meaning. In the next subsection we deal with examples where it is crucial that the variables take on values of either 0 or 1.

### 1.3.2 Capital Budgeting

Perhaps the easiest (from a pedagogical standpoint) integer program is the *capital budgeting problem*. The problem is to decide which projects to fund

given a constraint on available capital. The net present value is calculated for each project and the objective is to maximize the net present value of the sum of the selected projects subject to the constraint on available funding.

#### Parameters:

$n$  – the number of projects under consideration

$c_j$  – net present value of project  $j$

$a_j$  – capital required to fund project  $j$

$b$  – capital available for all projects

#### Variables:

$x_j$  – is equal to 1 if project  $j$  is funded and 0 if not

The integer linear program for the capital budgeting problem is

$$\max \sum_{j=1}^n c_j x_j \quad (1.18)$$

$$(CB) \quad \text{s.t.} \quad \sum_{j=1}^n a_j x_j \leq b \quad (1.19)$$

$$x_j \in \{0, 1\}, \quad j = 1, \dots, n. \quad (1.20)$$

The model formulation  $(CB)$  is due to Lorie and Savage [293] and is one of the first integer programming models studied. More complicated versions include multiple time periods and logical restrictions such as “if project  $j$  is undertaken then project  $k$  cannot be undertaken.” Bean, Noon, and Salton [48] solve a multiperiod version of this problem where the objective is to “divest” of assets for the Homart Development division of Sears in a manner so that the generated cash flow is smooth. The decision variables are  $x_{it} = 1$  if asset  $i$  is sold in year  $t$ . Bean, Noon, and Salton found that this model resulted in an additional profit of \$40 million dollars compared to traditional solution methods for the problem. The National Cancer Institute in Bethesda, MD uses an integer programming capital budgeting model to help make project funding decisions. See the paper by Hall et al. [222].

Problem formulation (*CB*) is also called the *knapsack problem* in the literature. Think of  $b$  as the capacity of the knapsack,  $a_j$  as the capacity required for packing item  $j$  and  $c_j$  as the utility of item  $j$ . The ratio of the objective function coefficient to the knapsack coefficient,  $c_j/a_j$ , is called the *bang-for-buck ratio*. It is a measure of the return on investment for project  $j$ . Later we study decomposition algorithms for solving large integer programs. The knapsack problem is often a component or subproblem of many larger more realistic problems and the decomposition algorithms require solving a knapsack problem as a subroutine. See the book by Martello and Toth [313] for solution algorithms for the knapsack problem and many of its variants.

### 1.3.3 Vehicle Routing

Mathematical programming models have been very important in reducing logistics costs by aiding the very complicated vehicle routing process. A problem that plays a prominent role in vehicle routing is the *generalized assignment problem*. The problem is to assign each of  $n$  tasks to  $m$  servers without exceeding the resource capacity of the servers.

**Parameters:**

$n$  – number of required tasks

$m$  – number of servers

$f_{ij}$  – cost of assigning task  $i$  to server  $j$

$b_j$  – units of resource available to server  $j$

$a_{ij}$  – units of server  $j$  resource required to perform task  $i$

**Variables:**

$x_{ij}$  – a binary variable which is equal to 1 if task  $i$  is assigned to server  $j$  and 0 if not

The integer linear program for the generalized assignment problem is

$$\min \sum_{i=1}^n \sum_{j=1}^m f_{ij} x_{ij} \quad (1.21)$$

$$(GAP) \quad \text{s.t.} \quad \sum_{j=1}^m x_{ij} = 1, \quad i = 1, \dots, n \quad (1.22)$$

$$\sum_{i=1}^n a_{ij} x_{ij} \leq b_j, \quad j = 1, \dots, m \quad (1.23)$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \quad (1.24)$$

The objective function (1.21) is to minimize the total assignment cost. Constraint (1.22) requires that each task is assigned a server. The requirement that the server capacity not be exceeded is given in (1.23).

One important application of this problem in the area of vehicle routing is to assign customers to trucks. Each truck  $j$  is a “server” with truck capacity  $b_j$  and  $a_{ij}$  units of truck capacity are used when customer  $i$  is assigned to truck  $j$ . Once customers are assigned to trucks it is necessary to route the trucks. This problem is the famous *traveling salesman problem*. For simplicity, assume a single truck with infinite capacity and the objective is to find a minimum cost route for the truck from the home depot, through  $n$  cities, and then back to the home depot.

#### Parameters:

$n$ — number of customers (or cities) to visit, not including the home depot indexed by  $i = 0$

$c_{ij}$ — cost of travel from customer location  $i$  to customer location  $j$

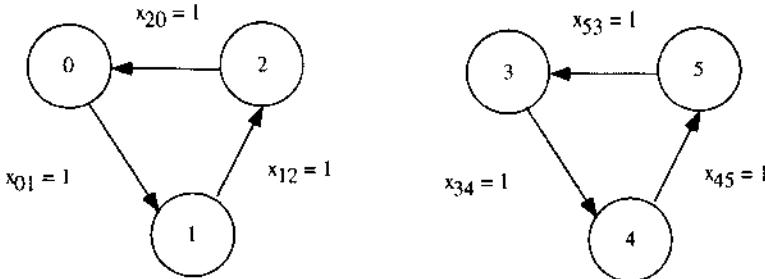
#### Variables:

$x_{ij}$ — a binary variable equal to 1 if the truck travels directly from customer location  $i$  to customer location  $j$

The integer linear program for the traveling salesman problem is

$$\min \sum_{i=0}^n \sum_{\substack{j=0 \\ i \neq j}}^n c_{ij} x_{ij} \quad (1.25)$$

$$(TSP) \quad \text{s.t.} \quad \sum_{\substack{i=0 \\ i \neq j}}^n x_{ij} = 1, \quad j = 0, \dots, n \quad (1.26)$$

**Figure 1.1** An Example of Subtours

$$\sum_{\substack{j=0 \\ i \neq j}}^n x_{ij} = 1, \quad i = 0, \dots, n \quad (1.27)$$

$$\sum_{\substack{(i,j) \in S \times S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad S \subseteq \{1, \dots, n\}, \quad 2 \leq |S| \leq n - 1 \quad (1.28)$$

$$x_{ij} \in \{0, 1\}, \quad i \neq j \quad (1.29)$$

The objective function (1.25) is to minimize total travel cost. The condition that the truck must visit each customer is given in constraint (1.26). Similarly, after visiting customer  $i$ , the truck must leave for another customer. This condition is given in (1.27). The truck must also leave and enter the depot which is indexed by 0.

The constraints in (1.28) are *subtour breaking constraints* and are not obvious at first reading. To illustrate their purpose assume  $n = 5$ . A feasible solution to the constraints (1.26) - (1.27) is illustrated in Figure 1.1. This solution is not a tour from the depot through every city. However, if for example,  $S = \{3, 4, 5\}$  then the solution illustrated in Figure 1.1 violates the subtour breaking constraint

$$x_{34} + x_{43} + x_{45} + x_{54} + x_{53} + x_{35} \leq 2.$$

The subtour breaking constraints are due to Dantzig, Fulkerson, and Johnson [114].

The generalized assignment problem and traveling salesmen problem were used by Fisher and Jaikumar [149] for routing trucks for Air Products and Chemicals resulting in logistical cost savings on the order of 13%, which is very significant since this is their major cost. The generalized assignment problem has

also been used for sales force allocation, see Zoltners and Sinha [479], facility location, see Ross and Soland [388] and snow removal routing, see Campbell and Langevin [83]. The traveling salesman problem has also been used in numerous applications. For more on this classic problem see Lawler et al. [281] and Reinelt [379].

### 1.3.4 Production Scheduling

Here we consider a very simple model called the *multiproduct lot size problem*. The problem is to determine a production schedule for  $N$  products over  $T$  time periods. Demand for each product must be met in each time period and stockouts are not permitted. There is a marginal cost of production and there is a marginal holding cost (usually based on the cost of capital) charged to the inventory held at the end of each time period. There is also a fixed cost of production (perhaps a machine setup) charged to each product if there is nonzero production in a period. Products compete for a scarce resource (machine capacity) in each time period.

#### Parameters:

$T$  – number of time periods

$N$  – number of products

$d_{it}$  – demand for product  $i$  in period  $t$

$f_{it}$  – fixed cost associated with production of product  $i$  in period  $t$

$h_{it}$  – marginal cost of holding one unit of product  $i$  in inventory at the end of period  $t$

$c_{it}$  – marginal production cost of one unit of product  $i$  in period  $t$

$g_t$  – production capacity available in period  $t$

$M_{it}$  – an upper bound on the production of product  $i$  in time period  $t$ , often called “big M” in the integer programming literature

#### Variables:

$x_{it}$  – units of product  $i$  produced in period  $t$

$I_{it}$  – units of product  $i$  held in inventory at the end of period  $t$

$y_{it}$  – a binary variable which is fixed to 1 if there is nonzero production of product  $i$  in period  $t$ , otherwise it is fixed to 0

The mixed integer linear program for the multiproduct dynamic lot size problem is

$$\min \quad \sum_{i=1}^N \sum_{t=1}^T (c_{it}x_{it} + h_{it}I_{it} + f_{it}y_{it}) \quad (1.30)$$

$$(MPDLS) \quad \text{s.t.} \quad \sum_{i=1}^N x_{it} \leq g_t, \quad t = 1, \dots, T \quad (1.31)$$

$$I_{i,t-1} + x_{it} - I_{it} = d_{it}, \quad i = 1, \dots, N, \quad t = 1, \dots, T \quad (1.32)$$

$$x_{it} - M_{it}y_{it} \leq 0, \quad i = 1, \dots, N, \quad t = 1, \dots, T \quad (1.33)$$

$$x_{it}, I_{it} \geq 0, \quad i = 1, \dots, N, \quad t = 1, \dots, T \quad (1.34)$$

$$y_{it} \in \{0, 1\}, \quad i = 1, \dots, N, \quad t = 1, \dots, T. \quad (1.35)$$

The objective function (1.30) is the minimization of the sum of the production, inventory holding and setup costs. Constraint (1.31) is a capacity constraint. It enforces the condition that the total production in each period  $t$  cannot exceed the production capacity  $g_t$  for that period. Constraint (1.32) is a *conservation of flow or sources and uses* requirement. It enforces the condition that for each product, in each time period, the beginning inventory for the period plus production in the period must equal demand plus ending inventory for the period. Constraint (1.33) is a *fixed charge or setup forcing constraint*. This constraint forces  $y_{it}$  to be nonzero if  $x_{it}$  is nonzero. Since each  $y_{it}$  is constrained to be 0 or 1 the only nonzero value is 1. Thus, if there is positive production of product  $i$  in period  $t$ , i.e.  $x_{it} > 0$  then  $y_{it} = 1$  and the fixed cost of  $f_{it}$  is charged. It is feasible to have  $x_{it} = 0$  and  $y_{it} = 1$  but since the objective function is a minimization this will not be the case since the fixed cost  $f_{it}$  is positive in any realistic setting.

The formulation (1.30) - (1.35) is a very simplified version of more realistic problems. In an actual industrial setting one might have backlogging of demand, multiple production lines, a fixed cost charged to product  $i$  only if it was not produced in a previous period, and multiple facilities with the output from one facility an input to another facility. Liberatore and Miller [288] apply a more complicated version of (MPDLS) to production planning at American Olean Tile Company. They report reduced costs from the model on the order of \$400,000 - \$700,000 per year. For other applications of this basic model

see Afentakis, Gavish, and Karmarkar [5], Barany, Van Roy, and Wolsey [34], Graves [206], Dzielinski and Gomory [135], Lasdon and Terjung [278], Manne [308], and many others. The multiproduct lot sizing problem is studied in greater length in Chapters 12 and 15.

### 1.3.5 Location Models

When the holder of a bank card pays his or her monthly bill the account of the issuer of the card is not credited until the customer's check clears. This elapsed time, often called *float time*, is a function of the number of days required for a check drawn on the customer bank  $j$  to clear at the card issuer bank  $i$ . Obviously, the issuer of the card would like to minimize float time. In order to minimize the float time the issuer could open a lock box account with a bank in the same city of every customer. The lock box operated by the bank is a post office box and the bank servicing the lock box collects the payment checks and transfers the funds to the corporation's main bank. The banks typically charge a fixed cost for the lock box and a variable per check fee. The cost of operating a lock box in the city of every customer would be prohibitive. The problem minimizing the sum of the cost of capital due to float and the cost of operating the lock boxes is the *lock box location problem*.

#### Parameters:

$m$  – number of customers to be assigned a lock box

$n$  – number of potential lock box sites

$c_{ij}$  – annual cost of capital associated with serving customer  $j$  from lock box  $i$

$f_i$  – annual fixed cost of operating a lock box at location  $i$

#### Variables:

$x_{ij}$  – a binary variable which is equal to 1 if customer  $j$  is assigned to lock box  $i$  and 0 if not

$y_i$  – a binary variable which is equal to 1 if the lock box at location  $i$  is opened and 0 if not

The integer linear program for the lock box location problem is

$$\min \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} + \sum_{i=1}^n f_i y_i \quad (1.36)$$

$$(LB) \quad \text{s.t.} \quad \sum_{i=1}^n x_{ij} = 1, \quad j = 1, \dots, m \quad (1.37)$$

$$x_{ij} - y_i \leq 0, \quad i = 1, \dots, n, \quad j = 1, \dots, m \quad (1.38)$$

$$x_{ij}, y_i \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \quad (1.39)$$

The objective (1.36) is to minimize the sum of the cost of capital plus the fixed cost of operating the lock boxes. The requirement that every customer be assigned a lock box is modeled by constraint (1.37). Constraints (1.38) are forcing constraints and play the same role as constraint set (1.33) in the dynamic lot size model. The importance of locating lock box accounts in order to minimize check clearing times is discussed in the November 23, 1974 issue of *Business Week*. Cornuéjols, Fisher, and Nemhauser [101] report computational experience with this model using actual bank data. See also Nauss and Markland [349] for a very thorough discussion of data gathering issues for generating the model parameters and computational results with real world problems.

The lock box model just described is also known as the *simple plant location problem*. In the simple plant location model,  $x_{ij} = 1$  if plant  $i$  serves customer  $j$ , 0 if not;  $c_{ij}$  is the cost of satisfying customer  $j$  demand from plant  $i$  and  $f_i$  is the fixed cost of operating (or opening) a plant at location  $i$ . It is also called the *uncapacitated plant location problem* because it is assumed that a plant has the capacity to service the entire demand of all customers assigned to the plant. A more realistic version of the problem is the *capacitated plant location problem*. The capacitated version is formulated below.

#### Parameters:

$m$  – number of customer locations

$n$  – number of potential plant locations

$c_{ij}$  – marginal cost of supplying one unit of customer  $j$  demand from plant  $i$

$f_i$  – fixed cost of operating (opening) a plant at location  $i$

$d_j$  – demand at customer location  $j$

$s_i$  – supply available at plant  $i$

**Variables:**

$x_{ij}$  – units of product sent from plant  $i$  to customer  $j$

$y_i$  – a binary variable which is equal to 1 if plant  $i$  is in operation or built and 0 if not

The mixed integer linear program for the capacitated plant location problem is

$$\min \sum_{i=1}^n \sum_{j=1}^m c_{ij}x_{ij} + \sum_{i=1}^m f_i y_i \quad (1.40)$$

$$(CPL) \quad \text{s.t.} \quad \sum_{i=1}^n x_{ij} = d_j, \quad j = 1, \dots, m \quad (1.41)$$

$$\sum_{j=1}^m x_{ij} - s_i y_i \leq 0, \quad i = 1, \dots, n \quad (1.42)$$

$$x_{ij} \geq 0, \quad i = 1, \dots, n, \quad j = 1, \dots, m \quad (1.43)$$

$$y_i \in \{0, 1\}, \quad i = 1, \dots, n. \quad (1.44)$$

Constraint (1.42) models the requirement that plant  $i$  cannot supply more than its capacity  $s_i$ , hence the name capacitated plant location. It also forces the fixed cost to be incurred if the plant is in operation. A more complicated version of this problem was implemented for Hunt Wesson Food, Inc. by Geoffrion and Graves [175]. The problem had 11,854 rows, 23,513 continuous variables, and 727 0-1 integer variables. For a text on location models and methodology see Mirchandani and Francis [336].

The capacitated plant location problem is a generalization of the famous *transportation problem*. In the transportation problem, the location aspect of the capacitated plant location problem is solved, and the problem is to find a minimum cost plan for satisfying demand without exceeding plant capacities. A generic formulation (*TLP*), is given below.

$$\min \sum_{i=1}^n \sum_{j=1}^m c_{ij}x_{ij} \quad (1.45)$$

$$(TLP) \quad \text{s.t.} \quad \sum_{i=1}^n x_{ij} = d_j, \quad j = 1, \dots, m \quad (1.46)$$

$$- \sum_{j=1}^m x_{ij} = -s_i, \quad i = 1, \dots, n \quad (1.47)$$

$$x_{ij} \geq 0, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \quad (1.48)$$

The variables and parameters in this formulation have the same meaning as in the capacitated plant location problem. A transportation linear program is an example of a *network flow linear program*. This is because of a natural network interpretation where the supply and demand locations are vertices of a directed graph and the flow from location  $i$  to demand center  $j$  is represented by a directed arc  $(i, j)$ . See Appendix C on graphs and networks. We revisit this problem later in the Chapter 14. A useful property of  $(TLP)$  is that there is always a feasible solution if total supply is equal to or greater than total demand.

### 1.3.6 Crew Scheduling

An extremely important problem faced by every airline is the *crew scheduling problem*. This is the problem of assigning crews to flights. Virtually all of the major airlines solve this problem using mathematical programming. The approach taken is to first identify the demand – that is, the flight requirements. For example, flight 107 from Chicago to San Antonio must be staffed every afternoon. Next, generate a large set of tours that a flight crew can take. For example, a tour might be flight 107 from Chicago to San Antonio, flight 189 from San Antonio to Denver and flight 866 from Denver to Chicago. A feasible tour depends upon the time that the flights are taken, FAA regulations etc. The cost of the tour depends upon its length, the amount of dead heading, etc. Once a large set of tours is generated, an integer linear programming model is used to find a subset of tours which cover the flight requirements.

**Parameters:**

$m$  – number of flights to cover

$n$  – number of tours generated

$a_{ij}$  – 1 if tour  $j$  includes flight  $i$  and 0 otherwise

$c_j$  – cost of tour  $j$

**Variables:**

$x_j$ — a binary variable which is 1 if tour  $j$  is used and 0 otherwise

The integer linear program for the crew scheduling problem is

$$\min \quad \sum_{j=1}^n c_j x_j \quad (1.49)$$

$$(CS) \quad \text{s.t.} \quad \sum_{j=1}^n a_{ij} x_j = 1, \quad i = 1, \dots, m \quad (1.50)$$

$$x_j \in \{0, 1\}, \quad j = 1, \dots, n \quad (1.51)$$

Model  $(CS)$  can be very large in practice. For example, American Airlines has over 8,000 planes and 16,000 flight attendants to schedule. They estimate that their mathematical programming based system TRIP (trip evaluation and improvement program) saves on the order of \$20 million per year. See Anbil, Tanga, and Johnson [9].

## 1.4 SPECIAL STRUCTURE

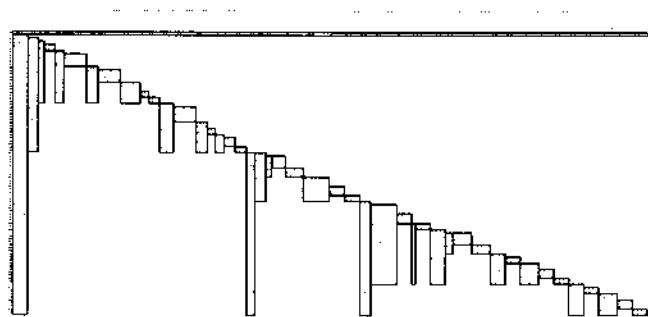
If one were to look at a picture of the pattern of nonzero elements in the constraint matrix of a real problem it would not look like a random shotgun blast. All real world linear and integer linear programs have special structure. First, they are *sparse*, that is the density of nonzero elements is very small. The density of nonzero elements in an  $m \times n$  matrix  $A$  is the number of nonzero elements in  $A$  divided by  $mn$ . Refer to Figures 1.2-1.3 and Tables 1.3-1.5. Figure 1.2 is the constraint matrix for a communications network design problem from Bellcore. Figure 1.3 is the same problem with the variables and rows in different matrix order. These figures are taken from Menon [332]. In these figures we have “blocked off” the areas of the constraint matrices where nonzero elements appear. Table 1.3 is the constraint matrix for a three task, two server generalized assignment problem. Table 1.4 is the constraint matrix for a three plant, five customer lock box or simple plant location problem and Table 1.5 is the same problem with the variables and rows in different matrix order.

Not only are these problems very sparse but the nonzero elements have a pattern to them. They are not random! The constraint matrices in Tables 1.3 and 1.4

**Figure 1.2** Communications Network Design



**Figure 1.3** Communications Network Design - Modified Matrix Order



**Table 1.3** Block Angular Structure for Generalized Assignment Problem

$x_{11}$	$x_{21}$	$x_{31}$	$x_{12}$	$x_{22}$	$x_{32}$
1			1		
	1			1	
		1			1
6	7	9		8	5
				6	

**Table 1.4** Block Angular Structure for Simple Plant Location

exhibit *block angular* structure. In the generalized assignment problem if we remove the constraints that each task must be assigned a server, the problem decomposes into independent blocks, one for each server. The constraints that link the independent block constraints together are called *coupling constraints*. In the simple plant location problem represented in Table 1.4 the coupling constraints are the demand constraints. Without the demand constraints the problem separates into a block of setup forcing constraints, one for each plant.

The constraints which form the individual blocks have interesting structure. For example, in the generalized assignment problem the block constraints are

**Table 1.5** Dual Angular Structure for Simple Plant Location

$x_{11}$	$x_{21}$	$x_{31}$	$x_{12}$	$x_{22}$	$x_{32}$	$x_{13}$	$x_{23}$	$x_{33}$	$x_{14}$	$x_{24}$	$x_{34}$	$x_{15}$	$x_{25}$	$x_{35}$	$y_1$	$y_2$	$y_3$
1	1	1													-1		
	1														-1		
		1													-1		
			1	1	1										-1		
				1											-1		
					1										-1		
						1	1	1							-1		
							1								-1		
								1							-1		
									1	1	1				-1		
										1					-1		
											1				-1		
												1	1	1			
												1			-1		
													1		-1		
														1		-1	

knapsack problems for which special algorithms are available. Later in Chapter 11 we show how each of the block problems in the simple plant location problem are easily optimized using a specialized algorithm.

Recognizing the special structure present depends on the ordering of the variables and constraints. Table 1.5 is the same constraint matrix as Table 1.4 but with the variables and rows in different order. Notice how different it looks! The structure exhibited in Table 1.5 is called *dual angular*. In this case there are *coupling variables*. In the case of simple plant location the coupling variables are the fixed charged variables  $y_i$ . If these variables are fixed at 0/1 the problem decomposes into an independent block for each customer. In Chapter 10 a very simple algorithm is given for optimizing each of the block problems after the  $y_i$  variables are fixed at zero or one.

When solving extremely large problems it is crucial to take advantage of special structure. Indeed, Parts IV and V of the book are devoted to algorithms for taking advantage of special structure. It is important to note that it is not always obvious how to order a matrix and take advantage of the special structure and this is as much an art as a science. For example, consider the two

different orderings in Figures 1.2 and 1.3. It is not clear which block structure is more amenable to special purpose algorithms. How to order the constraint matrix and identify blocks for special structure algorithms is a topic of ongoing research. See Menon [332].

## 1.5 LINEAR AND INTEGER LINEAR PROGRAMMING CODES

Numerous commercial codes exist for optimizing linear and integer linear programs. In this book we make reference to, or make use of, several of the leading ones including the commercial codes CPLEX from CPLEX a division of ILOG. LINDO from LINDO Systems, Inc., OSL from IBM and the research codes MINTO developed by Nemhauser, Savelsbergh and Sigismondi [353] and OB1 developed by Lustig et al. [302]. Numerous other codes exist. See *Optimization Software Guide* by Moré and Wright [346]. Additional sources (including online) for software and test problems are provided in Appendix D. Below is a printout from the LINDO software package for the Pittsburgh Steel example used in Section 1.3. It is a good illustration of the information that a typical linear programming code provides the user.

```

MIN      0.03 P1 + 0.06449999 P2 + 0.06499999 F1 + 0.061 F2 + 0.1 A1
        + 0.13 A2 + 0.119 A3 + 0.08 CB + 0.021 S1 + 0.02 S2 + 0.0195 S3
SUBJECT TO
 2)   CB <=    20
 3)   S1 <=   200
 4)   S2 <=   200
 5)   S3 <=   200
 6)   0.04 P1 + 0.15 CB + 0.004 S1 + 0.001 S2 + 0.001 S3 >=   60
 7)   0.04 P1 + 0.15 CB + 0.004 S1 + 0.001 S2 + 0.001 S3 <=   70
 8)   0.1 P2 + 0.2 A2 + 0.08 A3 >=     6
 9)   0.1 P2 + 0.2 A2 + 0.08 A3 <=     9
10)   0.008999999 P1 + 0.045 P2 + 0.6 A1 + 0.09 A2 + 0.33 A3
        + 0.008999999 S1 + 0.003 S2 + 0.003 S3 >=    27
11)   0.008999999 P1 + 0.045 P2 + 0.6 A1 + 0.09 A2 + 0.33 A3
        + 0.008999999 S1 + 0.003 S2 + 0.003 S3 <=    33
12)   0.0225 P1 + 0.15 P2 + 0.45 F1 + 0.42 F2 + 0.18 A1 + 0.3 A2
        + 0.25 A3 + 0.3 CB >=    54
13)   0.0225 P1 + 0.15 P2 + 0.45 F1 + 0.42 F2 + 0.18 A1 + 0.3 A2
        + 0.25 A3 + 0.3 CB <=    60
14)   P1 + P2 + F1 + F2 + A1 + A2 + A3 + CB
        + S1 + S2 + S3 =    2000
END

```

: go  
 LP OPTIMUM FOUND AT STEP 10

## OBJECTIVE FUNCTION VALUE

1) 59.55629

VARIABLE	VALUE	REDUCED COST
P1	1474.264100	.000000
P2	60.000000	.000000
F1	.000000	.001036
F2	22.062050	.000000
A1	14.238862	.000000
A2	.000000	.020503
A3	.000000	.019926
CB	.000000	.003357
S1	200.000000	.000000
S2	29.434932	.000000
S3	200.000000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	20.000000	.000000
3)	.000000	.000177
4)	170.565100	.000000
5)	.000000	.000500
6)	-.000000	-.183329
7)	10.000000	.000000
8)	-.000000	-.254731
9)	3.000000	.000000
10)	-.000000	-.104521
11)	6.000000	.000000
12)	-.000000	-.098802
13)	6.000000	.000000
14)	.000000	-.019503

NO. ITERATIONS= 10

DO RANGE(SENSITIVITY) ANALYSIS?

?y

RANGES IN WHICH THE BASIS IS UNCHANGED:

OBJ COEFFICIENT RANGES

VARIABLE	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
P1	.030000	.000876	.002302
P2	.064500	.010252	.025473
F1	.065000	INFINITY	.001036
F2	.061000	.000967	.041639
A1	.100000	.040512	.019176
A2	.130000	INFINITY	.020503
A3	.119000	INFINITY	.019926
CB	.080000	INFINITY	.003357
S1	.021000	.000177	INFINITY
S2	.020000	.007613	.000192
S3	.019500	.000500	INFINITY

RIGHTHOOK SIDE RANGES			
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	20.000000	INFINITY	20.000000
3	200.000000	31.928833	185.016320
4	200.000000	INFINITY	170.565100
5	200.000000	29.434932	170.565100
6	60.000000	1.222390	7.083321
7	70.000000	INFINITY	10.000000
8	6.000000	3.000000	6.000000
9	9.000000	INFINITY	3.000000
10	27.000000	6.000000	8.520192
11	33.000000	INFINITY	6.000000
12	54.000000	6.000000	9.297775
13	60.000000	INFINITY	6.000000
14	2000.000000	166.078600	28.660690

The output report is quite extensive. In addition to the optimal solution value for the decision variables, there is other information such as a *reduced cost* associated with each variable and a *sensitivity analysis* report with information on the allowable increase and decrease for the right hand sides and objective function coefficients. All of these terms are explained in Chapters 2 and 3. In the LINDO computer printout the inequality symbols of  $\leq$  and  $\geq$  are represented by  $\leq$  and  $\geq$ , respectively. Also, the constraints are numbered beginning with constraint 2).

## 1.6 OTHER DIRECTIONS

We choose to emphasize the theory and algorithms for solving large scale linear and integer linear optimization. The area of mathematical optimization is vast and readers preferring a different emphasis may wish to consider the following references.

**Combinatorial Optimization and Network Emphasis:** The reader interested in textbooks with all, or considerable emphasis, on network flow models and algorithms should consult *Network Flows Theory, Algorithms, and Applications* by Ahuja, Magnanti, and Orlin [7] and *Linear Programming and Network Flows* by Bazaraa, Jarvis, and Sherali [43]. For emphasis on combinatorial optimization we recommend *Combinatorial Optimization Networks and Matroids* by Lawler [280], *Geometric Algorithms and Combinatorial Optimization* by Grötschel, Lovász and Schrijver [212] and *Combinatorial Optimization Algorithms and Complexity* by Papadimitriou and Steiglitz [368]. The text by Grötschel, Lovász and Schrijver is particularly thorough in its coverage and emphasis on the role of the ellipsoid algorithm in combinatorial optimization.

**Modeling Emphasis:** Readers wishing books with more emphasis on mathematical modeling and applications than in this text should consult *An Introduction to Management Science* by Anderson, Sweeney, and Williams [12], *Management Science Modeling Analysis and Interpretation* by Camm and Evans [82], *Introductory Management Science* by Eppen, Gould, and Schmidt [141], *AMPL A Modeling Language for Mathematical Programming* by Fourer, Gay, and Kernighan [158], *LINDO An Optimization Modeling System* by Schrage [402], and *Model Building in Mathematical Programming* by Williams [451].

**Integer Programming:** Much of this book is devoted to integer linear programming. Other books which also put considerable emphasis on integer linear programming include *Integer Programming* by Garfinkle and Nemhauser [167], *Integer and Combinatorial Optimization* by Nemhauser and Wolsey [350], *Discrete Optimization* by Parker and Rardin [369], *Foundations of Integer Programming* by Salkin and Mathur [394], and *Theory of Linear and Integer Programming* by Schrijver [403].

**Interior Point Algorithms:** Recent linear programming books with considerable emphasis on interior point algorithms include *Linear Optimization and Extensions* by Fang and Puthenpura [146], *Theory and Algorithms for Linear Optimization* by Roos, Terlaky, and Vial [383], *Linear Programming A Modern Integrated Analysis* by Saigal, [393] and *Interior Point Algorithms Theory and Analysis* by Ye [469]. See also *Interior Point Approach to Linear, Quadratic and Convex Programming Algorithms and Complexity* by den Hertog [227].

**Introductory LP Texts:** There are numerous excellent textbooks on the market devoted almost exclusively to linear programming with little or no integer programming. Examples include *Linear Programming and Network Flows* by Bazaraa, Jarvis, and Sherali [43], *Linear Programming* by Chvátal [93], *Linear Programming* by Gass [168], *Linear Programming* by Hadley [220], *Linear Programming* by Karloff [259], and *Linear Programming Foundations and Extensions* by Vanderbei [443]. The book *Linear Optimization and Extensions* by Padberg [361] has a strong emphasis on the geometric and polyhedral aspects of linear programming. See Birge and Louveaux [53] for a good text on stochastic linear programming. *Linear Programming and Extensions* by George Dantzig [108] remains a classic. It has recently been updated as *Linear Programming 1: Introduction* by Dantzig and Thapa [117].

## 1.7 EXERCISES

- 1.1 Modify the formulation of the Pittsburgh Steel Company linear program so that the one ton requirement can be varied and only one number in the formulation will change.
- 1.2 Show that any integer program where all the decision variables are bounded can be converted into a binary integer program.
- 1.3 **Cash Flow Matching:** There are a number of important applications in which a stream of cash flows must be generated over a planning horizon. One example is a personal injury lawsuit where the plaintiff must be compensated for future medical expenses and/or lost wages. In such cases both parties often agree on an initial lump sum which is “equivalent” to the sequence of cash flows. In making such an agreement the plaintiff argues for a very low interest rate to use in calculating the present value of the income stream.

Conversely, the counsel for the defense will argue for a smaller lump sum (i.e., higher interest rate). The argument to the plaintiff might proceed as follows: “I can show you a portfolio such that my smaller lump sum will generate an income stream matching your required income stream.” Obviously the plaintiff is inclined to accept such an argument only if the proposed portfolio is of very low risk.

A second example is pension fund planning. Here a corporation or union has an obligation to meet the cash requirements of a pension fund over some planning horizon. It would like to do so by constructing a minimum

cost, low risk portfolio which generates an income stream to match the required income stream.

Jim E. Hoffa runs a small consulting company which specializes in constructing portfolios for pension funds. He has recently been hired by a local union to construct its portfolio. The actuaries hired by the union have determined the cash requirements for the next 10 years. See Table 1.6. Hoffa is very much in favor of constructing a low risk portfolio be-

**Table 1.6 Cash Flow Requirements**

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
Cash ( $\times 1000$ )	10	11	12	14	15	17	19	20	22	24

cause of the adverse consequences that might arise if the portfolio cannot meet the required cash needs of the union. Thus, Hoffa is considering four types of low risk investments: an insured money market fund paying an annual interest rate of 5%, and three types of AAA rated bonds. The data on the bonds are given in Table 1.7.

**Table 1.7 Bond Data**

Bond	Current Cost	Coupon Return	Years to Maturity	Repayment At Mat.
One	\$921	\$40	3	\$1000
Two	\$810	\$50	5	\$1000
Three	\$783	\$0.0	5	\$1000

Assume that all transactions occur on January 1 of each year. For example, if one bond of Bond Two is purchased in 1991 the following transactions would occur. On January 1, 1991 \$810 is paid for the bond. On January 1, 1992-1995 a \$50 coupon return is received. On January 1, 1996 a \$50 coupon return plus the \$1000 face value are received. Note that Bond Three is a zero coupon bond and does not pay any interest until maturity.

Bonds can be purchased in any of the years 1990 through 1999. In order to purchase bonds it may be necessary to borrow money. Assume that money can be borrowed at an annual rate of 13%, again with transactions taking place on January 1 of each year. You cannot borrow money in 1999. Thus, if \$1000 is borrowed on January 1, 1993 then \$1130 must be paid (interest plus principal) on January 1, 1994. The money market works in a similar fashion. Hoffa's objective is to construct a portfolio to meet the

cash requirements given in Figure 1.6 using the minimum initial lump sum of money.

In answering the following questions you may assume that any fraction of a bond can be purchased, i.e., an integer number of bonds need not be purchased.

- a. Comment on the following: "The way to minimize the initial lump sum required is to invest, as needed, *only* in the investment which has the largest rate of return."
- b. Comment on the following: "If the rate of return on the money market fund exceeds the rate of return on all of the bonds then there is an optimal solution in which the money market fund is the only investment used."
- c. Formulate this problem (minimizing the initial money requirements while meeting cash flows) as a linear program.
- d. Comment on the following: "The linear programming model will be feasible for any nonnegative set of cash requirements over the 10 year planning horizon."
- e. Comment on the following: "If the borrowing rate were less than either the money market rate or the rate of return on one of the bonds the problem would be unbounded."
- f. Comment on the following: "If a zero coupon bond has a rate of return less than the rate of return on a bond which pays coupons, and both bonds have equal length to maturity, then there will always be an optimal solution without the zero coupon bond in the portfolio."
- g. Suppose borrowing is no longer permitted. Now comment on the following: "In every optimal solution the amount of money invested in the money market in the final period (1999) must be zero since this money is never available for future use in the model and must have cost us something to acquire it."
- h. Comment on the following: "If a zero coupon bond has  $k$  years to maturity then there exists an optimal portfolio in which this bond is not purchased  $k$  years before the end of the planning horizon."
- i. As a plaintiff, or union member planning retirement, what worry might you have in agreeing by the solution given to the linear program for this problem?
- j. Comment on the following: "If the borrowing rate is strictly greater than the return on any of the bonds or the money market, then it is never optimal to borrow in period 1." If the borrowing rate is strictly

greater than the return on any bond or the money market is it ever optimal to borrow? Why or why not?

- 1.4 Roger Gibson is in the process of constructing a healthy diet for Clay Henry, Jr., the Mayor<sup>1</sup> of Lajitas, Texas. The Mayor consumes oats, beans, and beer. Roger would like to meet certain nutritional requirements for the Mayor at minimum cost. In particular, Roger would like Clay Henry, Jr. to have at most 3000 calories per day, at most 12 grams of fat, at least 5 grams of protein and at least 150 grams of carbohydrates. The amount of carbohydrates, fat and protein (in grams) in oats, beans, and beer are given in Table 1.8. Assume that the Mayor always finishes every beer he starts. Formulate an integer linear program for constructing a minimum cost diet for Clay Henry, Jr.

**Table 1.8** Diet Information for Clay Henry, Jr.

	Calories	Carbohydrates	Fat	Protein	Cost
Beans (1 cup)	240	54	10	12	\$0.30
Oats (1 cup)	300	54	6	10	\$0.45
Beer (12 ounces)	110	8.3	0.0	0.7	\$1.50

- 1.5 Prove that if total supply is greater than or equal to total demand, then there is a feasible solution to the transportation linear program (*TLP*).

---

<sup>1</sup>The Mayor of Lajitas, Texas is a goat.

---

**PART II**

---

**THEORY**

## LINEAR SYSTEMS AND PROJECTION

### 2.1 INTRODUCTION

Finding solutions to systems of linear equalities and inequalities is an incredibly important and fascinating problem from both an *applied* and a *theoretical* standpoint. Work in this area dates back at least as early as 2000 B.C. Virtually every linear algebra textbook has a result such as:

**Proposition 2.1** *Every square matrix  $A$  has precisely one of the following two properties:*

1.  *$A$  is nonsingular and for every  $b$ , the system  $Ax = b$  has a unique solution.*
2.  *$A$  is singular and for every  $b$ , the system  $Ax = b$  has either no solution or infinitely many solutions.*

In this chapter we extend this result to inequality systems  $Ax \geq b$  without making any restrictions on  $A$  being a square matrix. Two important questions are:

1. Under what conditions does the system  $Ax \geq b$  have a solution?
2. How can we find a solution to the system  $Ax \geq b$ ?

Our methods are constructive. That is, we determine existence of a solution by either finding a solution or showing that one does not exist. The method we

use is *projection*. This method goes back to Fourier [159] and Motzkin [347]. Dines [125] also discovered the method of Fourier. More recently, Dantzig and Eaves [109], Duffin [133], Kuhn [273], and Williams [453] have applied the ideas of Fourier and Motzkin to linear programming. Our development is based on their work, especially Williams. In Section 2.2 we describe the well known Gaussian elimination algorithm for linear equalities as a projection algorithm. In Section 2.3 the idea of projection is extended to solve systems of linear inequalities. Two applications of projection, including linear programming, are given in Section 2.4. In Section 2.5 we show how many of the classic “theorems of the alternative” are derived through projection. The important concept of linear programming duality is introduced in Section 2.6. It is motivated through projection. The concept of complementary slackness is introduced in Section 2.7. Section 2.8 is devoted to sensitivity analysis of changes in the right hand side vector. Concluding remarks are made in Section 2.9. Exercises are provided in Section 2.10.

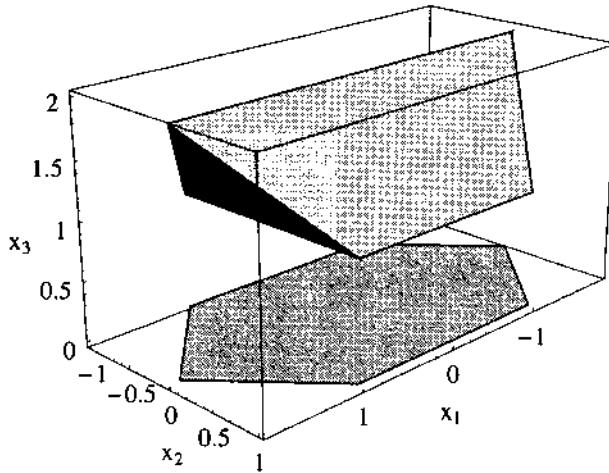
Before proceeding, the reader not familiar with basic polyhedral theory should read Sections A.2 and A.3 in Appendix A. Several important definitions from Appendix A are repeated here. First, we give a precise meaning to the concept of projection. The intersection of a finite number of half-spaces is a *polyhedron*. Consider the polyhedron,  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ . Throughout the text,  $\mathbb{R}$  denotes the set of real numbers. The *projection* of the polyhedron  $P$  into the subspace of the  $y$  variables is

$$\text{proj}_y(P) := \{y \in \mathbb{R}^{n_2} \mid \text{there exists } x \in \mathbb{R}^{n_1} \text{ such that } (x, y) \in P\}.$$

The symbol  $:=$  is instead of  $=$  in a definitional equation. In Figure 2.1 we illustrate the projection of a three dimensional polyhedron defined by five inequalities into the  $(x_1, x_2)$  plane. That is, the polyhedron that results from projecting out variable  $x_3$ . A formal proof is given later that the projection of a polyhedron is again a polyhedron.

## 2.2 PROJECTION FOR EQUALITY SYSTEMS: GAUSSIAN ELIMINATION

The *key idea* used throughout this entire chapter is projection. This is a technique everyone learned in high school to solve systems of equalities. The procedure takes a problem with  $n$  variables and through substitution “projects out” a variable resulting in a system with  $n - 1$  of the variables and one less constraint. This process is repeated until only one variable remains. The problem with one variable is trivial to solve.

**Figure 2.1** Projection of a Polyhedron**Example 2.2**

$$3x_1 - 6x_2 + 4x_3 = 1 \quad (E1)$$

$$-x_1 + 3x_2 - 2x_3 = 3 \quad (E2)$$

$$x_1 - 4x_2 + x_3 = 0 \quad (E3)$$

*Project out variable  $x_1$ . Do this by multiplying (E1) by  $1/3$  and then adding  $(1/3)(E1)$  to (E2) and adding  $-(1/3)(E1)$  to (E3). This gives the “equivalent” system.*

$$1x_2 - (2/3)x_3 = 10/3 \quad (1/3)(E1) + (E2)$$

$$-2x_2 - (1/3)x_3 = -1/3 \quad -(1/3)(E1) + (E3)$$

*Next project out variable  $x_2$ .*

$$-(5/3)x_3 = 19/3 \quad (1/3)(E1) + 2(E2) + (E3)$$

*The solution to this equation is  $x_3 = -19/5$ . Then solving for  $x_2$  gives  $x_2 = 12/15$ . Finally,  $x_1 = 105/15$ .*

It is crucial to understand that each time a variable is projected out there is a feasible solution to the resulting system if and only if there was a feasible

solution to the original system. In general, given a system

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \quad (2.1)$$

$$a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n = b_k, \quad k = 2, \dots, m \quad (2.2)$$

an equivalent system is (assume without loss that  $a_{11} \neq 0$ .)

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \cdots + \frac{a_{1n}}{a_{11}}x_n = b_1/a_{11} \quad (2.3)$$

$$\left( a_{k2} - a_{k1}\frac{a_{12}}{a_{11}} \right)x_2 + \cdots + \left( a_{kn} - a_{k1}\frac{a_{1n}}{a_{11}} \right)x_n = \left( b_k - a_{k1}\frac{b_1}{a_{11}} \right), \\ k = 2, \dots, m \quad (2.4)$$

Notice that (2.4) is the system of equations that results from projecting out variable  $x_1$  from the system (2.1)-(2.2) using the substitution

$$x_1 = (b_1/a_{11}) - (a_{12}/a_{11})x_2 - \cdots - (a_{1n}/a_{11})x_n.$$

What we are calling projection is most commonly known as *Gaussian elimination*. The numerical aspects of Gaussian elimination are studied in detail later in Chapter 13.

**Proposition 2.3** *The projection of the feasible region in  $\mathbb{R}^n$  defined by (2.1)-(2.2) into the  $(x_2, \dots, x_n)$  subspace is defined by (2.4) and there is a feasible solution to (2.1)-(2.2) if and only if there is a feasible solution to (2.4).*

Proposition 2.3 is very trivial and because everyone has assimilated it so well we lose track of its significance. We include Proposition 2.3 because the idea of projecting out variables in order to obtain an easier system is later applied to linear inequalities and integer linear equalities and inequalities. Proposition 2.3 illustrates exactly the kind of result we want for more general systems.

#### Example 2.4

$$x_1 + 2x_2 = 6$$

$$2x_1 + 4x_2 = 20$$

*Substituting out or projecting out  $x_1 = 6 - 2x_2$  gives  $0x_2 = 8$  which is obviously a false statement about real numbers. Because projection has led to this “inconsistency” we conclude that the original system has no solution. This substitution corresponds to multiplying the first equation by  $-2$  and adding the result to the second equation. Example 2.2 and 2.4 illustrate the following proposition.*

**Proposition 2.5** Either the system

$$Ax = b$$

has a solution, or the system

$$A^T u = 0, \quad b^T u \neq 0$$

has a solution, but not both.

Proposition 2.5 is proved using a stronger result in the next section. In these examples projection either found a solution or showed that one did not exist. This is something we would like do in general for both equalities and inequalities. Gaussian elimination is not valid for inequalities because multiplication by a negative number changes the direction of the inequality. However, there is a way to project out variables when inequalities are present.

## 2.3 PROJECTION FOR INEQUALITY SYSTEMS: FOURIER-MOTZKIN ELIMINATION

Consider the polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  where  $A$  is an  $m$  by  $n$  matrix of real numbers and  $b \in \mathbb{R}^m$ . The objective is to determine whether or not the system  $Ax \geq b$  has a solution. As with Gaussian elimination, the basic idea is to project out variables until we reach a system of inequalities in one variable that is obviously either feasible (therefore  $Ax \geq b$  has a solution) or infeasible (therefore  $Ax \geq b$  does not have a solution). We show how to project out a single variable. This process is then repeated until there is a system with only one variable. Since each row of the system  $Ax \geq b$  can be multiplied by a positive scalar without changing  $P$ , assume without loss that the elements in the first column of the matrix are  $0, 1, -1$ . Then (possibly after reordering the rows) the system is

$$x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i, \quad i = 1, \dots, m_1 \quad (2.5)$$

$$-x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i, \quad i = m_1 + 1, \dots, m_1 + m_2 \quad (2.6)$$

$$a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i, \quad i = m_1 + m_2 + 1, \dots, m \quad (2.7)$$

Every solution to this system of equations satisfies

$$x_1 \geq b_i - a_{i2}x_2 - \cdots - a_{in}x_n, \quad i = 1, \dots, m_1 \quad (2.8)$$

$$x_1 \leq a_{k2}x_2 + \cdots + a_{kn}x_n - b_k, \quad k = m_1 + 1, \dots, m_1 + m_2. \quad (2.9)$$

Combining (2.8) and (2.9) gives

$$b_i - a_{i2}x_2 - \cdots - a_{in}x_n \leq x_1 \leq a_{k2}x_2 + \cdots + a_{kn}x_n - b_k. \quad (2.10)$$

for all  $i = 1, \dots, m_1$  and all  $k = m_1 + 1, \dots, m_1 + m_2$ . Thus, any solution to (2.5) - (2.7) is also a solution to (2.11) - (2.12).

$$(a_{k2} + a_{i2})x_2 + \cdots + (a_{kn} + a_{in})x_n \geq (b_i + b_k), \quad i = 1, \dots, m_1, \\ k = m_1 + 1, \dots, m_1 + m_2 \quad (2.11)$$

$$a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i, \quad i = m_1 + m_2 + 1, \dots, m \quad (2.12)$$

This method of eliminating variable  $x_1$  is called *Fourier-Motzkin elimination*. See Fourier [159] and Motzkin [347]. Throughout the rest of the book, whenever we use the term “projection,” or the phrase “eliminate a variable(s),” for variables in a system of linear inequalities, we are referring to Fourier-Motzkin elimination.

Denote the polyhedron defined by system (2.11) - (2.12) by  $\text{proj}'_{(x_2, \dots, x_n)}(P)$ . In order for projection to be a valid procedure it must be true that  $Ax \geq b$  has a solution if and only if (2.11) - (2.12) has a solution. That is, by projecting out a variable at each step we do not create or lose any solutions. Stated another way, it must be the case that  $\text{proj}'_{(x_2, \dots, x_n)}(P) = \text{proj}_{(x_2, \dots, x_n)}(P)$  where

$$\text{proj}_{(x_2, \dots, x_n)}(P) = \{(x_2, \dots, x_n) \mid \text{there exists } (x_1, \dots, x_n) \text{ such that } Ax \geq b\}.$$

By definition,  $\text{proj}'_{(x_2, \dots, x_n)}(P)$  is the *projection* of the polyhedron  $P = \{x \mid Ax \geq b\}$  into the subspace defined by the variables  $(x_2, \dots, x_n)$ . The next two Lemmas prove that the inequalities that result from eliminating  $x_1$  define the projection of the polyhedron defined by  $Ax \geq b$  into the  $(x_2, \dots, x_n)$  subspace.

**Lemma 2.6 (Solutions Not Lost)** *If  $P$  is not empty then  $\text{proj}'_{(x_2, \dots, x_n)}(P)$  is not empty and  $\text{proj}_{(x_2, \dots, x_n)}(P) \subseteq \text{proj}'_{(x_2, \dots, x_n)}(P)$ .*

**Proof:** If  $P$  is not empty then  $\text{proj}_{(x_2, \dots, x_n)}(P)$  is not empty. Let  $(\bar{x}_2, \dots, \bar{x}_n) \in \text{proj}_{(x_2, \dots, x_n)}(P)$ . By definition of projection, there exists  $(\bar{x}_1, \dots, \bar{x}_n)$  which is a solution to  $Ax \geq b$ . Then this solution satisfies (2.5) - (2.7) which satisfies (2.8) and (2.9) so  $(\bar{x}_2, \dots, \bar{x}_n)$  is a feasible solution to (2.11) - (2.12). Then  $(\bar{x}_2, \dots, \bar{x}_n) \in \text{proj}'_{(x_2, \dots, x_n)}(P)$  so  $\text{proj}'_{(x_2, \dots, x_n)}(P)$  is not empty and  $\text{proj}_{(x_2, \dots, x_n)} \subseteq \text{proj}'_{(x_2, \dots, x_n)}(P)$ .  $\square$

**Lemma 2.7 (Solutions Not Created)** *If  $\text{proj}'_{(x_2, \dots, x_n)}(P)$  is not empty then  $P$  is not empty and  $\text{proj}'_{(x_2, \dots, x_n)}(P) \subseteq \text{proj}_{(x_2, \dots, x_n)}(P)$ .*

**Proof:** If  $\text{proj}'_{(x_2, \dots, x_n)}(P)$  is not empty there exists  $(\bar{x}_2, \dots, \bar{x}_n)$  which satisfies (2.11) - (2.12). Let

$$\beta = \min \{ a_{k2}\bar{x}_2 + \dots + a_{kn}\bar{x}_n - b_k \mid k = m_1 + 1, \dots, m_1 + m_2 \}$$

and

$$\alpha = \max \{ b_i - a_{i2}\bar{x}_2 - \dots - a_{in}\bar{x}_n \mid i = 1, \dots, m_1 \}.$$

Then  $\alpha \leq \beta$  and completing the solution by using any

$$\alpha \leq \bar{x}_1 \leq \beta$$

gives a feasible solution to  $Ax \geq b$ . Then  $P$  is not empty and  $\text{proj}'_{(x_2, \dots, x_n)}(P) \subseteq \text{proj}_{(x_2, \dots, x_n)}(P)$ .  $\square$

It follows that  $P = \emptyset$  if and only if  $\text{proj}'_{(x_2, \dots, x_n)}(P) = \text{proj}_{(x_2, \dots, x_n)}(P) = \emptyset$ . This observation and Lemmas 2.6 and 2.7 imply the following proposition.

**Proposition 2.8** *If  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ , then  $\text{proj}_{(x_2, \dots, x_n)}(P)$  is defined by the inequalities (2.11)-(2.12).*

The projection of a polyhedron into any subspace of variables is easily found by successively applying this elimination process. This is illustrated in the following example.

### Example 2.9

$$3x_1 - 6x_2 + 6x_3 \geq 12 \quad (E1)$$

$$-x_1 + 3x_2 - 2x_3 \geq 3 \quad (E2)$$

$$x_1 - 4x_2 + x_3 \geq -15 \quad (E3)$$

$$-x_1 + 0x_2 + x_3 \geq -15 \quad (E4)$$

Multiply the first equation by (1/3) in order to get 1, -1 or 0 in the first column.

$$x_1 - 2x_2 + 2x_3 \geq 4 \quad (1/3)(E1)$$

$$-x_1 + 3x_2 - 2x_3 \geq 3 \quad (E2)$$

$$x_1 - 4x_2 + x_3 \geq -15 \quad (E3)$$

$$-x_1 + 0x_2 + x_3 \geq -15 \quad (E4)$$

Now project out the  $x_1$  variable by adding the first equation to the second and fourth equations, and the third equation to the second and fourth equation.

$$x_2 + 0x_3 \geq 7 \quad (1/3)(E1) + (E2)$$

$$-2x_2 + 3x_3 \geq -11 \quad (1/3)(E1) + (E4)$$

$$-x_2 - x_3 \geq -12 \quad (E2) + (E3)$$

$$-4x_2 + 2x_3 \geq -30 \quad (E3) + (E4)$$

Notice that the number of equations has not been reduced! This is not always the case. Refer back to Figure 2.1 which shows the projection of a three dimensional polyhedron defined by five inequalities. The projection into the  $(x_1, x_2)$  space results in a polyhedron defined by six inequalities. Now project out  $x_2$ . First make all coefficients +1, -1 or 0 in the  $x_2$  column and then add the first equation to the other three equations.

$$\begin{array}{rcl} x_2 & +0x_3 & \geq 7 \\ -x_2 & +(3/2)x_3 & \geq -11/2 \\ -x_2 & -x_3 & \geq -12 \\ -x_2 & +(1/2)x_3 & \geq -30/4 \end{array} \quad \begin{array}{l} (1/3)(E1) + (E2) \\ (1/6)(E1) + (1/2)(E4) \\ (E2) + (E3) \\ (1/4)(E3) + (1/4)(E4) \end{array}$$

The result is

$$\begin{array}{rcl} (3/2)x_3 & \geq & 3/2 \\ -x_3 & \geq & -5 \\ (1/2)x_3 & \geq & -1/2 \end{array} \quad \begin{array}{l} (1/2)(E1) + (E2) + (1/2)(E4) \\ (1/3)(E1) + 2(E2) + (E3) \\ (1/3)(E1) + (E2) + (1/4)(E3) + (1/4)(E4) \end{array}$$

This system is equivalent to  $1 \leq x_3 \leq 5$  which obviously has a solution. Therefore, by repeated application of Lemma 2.7 the original system has a solution.

Although the one variable system is easily solved, projecting out all the variables is also important. Project out  $x_3$  by adding  $(2/3)$  times the first equation to the second equation and 2 times the third equation to the second equation.

$$\begin{array}{rcl} 0 & \geq & -4 \\ 0 & \geq & -6 \end{array} \quad \begin{array}{l} (2/3)(E1) + (8/3)(E2) + (E3) + (1/3)(E4) \\ (E1) + 4(E2) + (3/2)(E3) + (1/2)(E4) \end{array}$$

These are obviously true statements about real numbers. Notice if we multiply equation (E1) by  $(2/3)$ , (E2) by  $(8/3)$ , (E3) by 1 and (E4) by  $(1/3)$  and then add we get the equation  $0x_1 + 0x_2 + 0x_3 \geq -4$ . This is called an *aggregate constraint* (see the following key observation 1) and it obviously has a solution.

If the right hand side of the original system had been  $(12, 3, 0, -15)$  instead of  $(12, 3, -15, -15)$  then projection yields the equations

$$\begin{array}{rcl} 0 & \geq & 11 \\ 0 & \geq & 33/2. \end{array}$$

These are obviously false statements about real numbers. If we multiply equation (E1) by  $(2/3)$ , (E2) by  $(8/3)$ , (E3) by 1 and (E4) by  $(1/3)$  and then add we get the equation  $0x_1 + 0x_2 + 0x_3 \geq 11$ . There is no solution to this aggregate constraint and so the original system has no solution. More on this later.

## KEY OBSERVATIONS

- Projection consists of two row operations - multiplying a row by a positive scalar and adding two rows together. These row operations are valid for inequalities and generate aggregate constraints. If  $\bar{u} \geq 0$ , then  $(\bar{u}^\top A)x \geq \bar{u}^\top b$  is an *aggregate constraint* of the system  $Ax \geq b$ . Consider the inequalities

$$2x_1 + 3x_2 \geq 6 \quad (2.13)$$

$$4x_1 + 2x_2 \geq 8 \quad (2.14)$$

The aggregate constraint  $8x_1 + 8x_2 \geq 20$  is obtained from this system by assigning a multiplier of 2 to inequality (2.13), a multiplier of 1 to inequality (2.14), and then adding. At each iteration of the projection method a new set of aggregate constraints is generated. The set of aggregate inequalities define the projected polyhedron.

- Projection applied to the system  $Ax \geq b$  generates aggregate constraints of the form  $(u^\top A)x \geq u^\top b$ . The multipliers used to generate these aggregate constraints are nonnegative solutions to the system  $A^\top u = 0$ . Refer back to Example 2.9 and recall that a  $\bar{u}^\top = (2/3, 8/3, 1, 1/3)$  gives the aggregate constraint  $0x_1 + 0x_2 + 0x_3 \geq -4$  and  $\bar{u}^\top = (1, 4, 3/2, 1/2)$  gives the aggregate constraint  $0x_1 + 0x_2 + 0x_3 \geq -6$ . Both of these  $u$  vectors are solutions to  $A^\top u = 0$  for the  $A$  matrix in the example.

$$\begin{bmatrix} 2/3 & 8/3 & 1 & 1/3 \end{bmatrix} \begin{bmatrix} 3 & -6 & 6 \\ -1 & 3 & -2 \\ 1 & -4 & 1 \\ -1 & 0 & 1 \end{bmatrix} = [0 \ 0 \ 0]$$

$$\begin{bmatrix} 1 & 4 & 3/2 & 1/2 \end{bmatrix} \begin{bmatrix} 3 & -6 & 6 \\ -1 & 3 & -2 \\ 1 & -4 & 1 \\ -1 & 0 & 1 \end{bmatrix} = [0 \ 0 \ 0]$$

Given the general system,

$$\begin{aligned} x_1 + a_{i2}x_2 + \cdots + a_{in}x_n &\geq b_i, \quad i = 1, \dots, m_1 \\ -x_1 + a_{i2}x_2 + \cdots + a_{in}x_n &\geq b_i, \quad i = m_1 + 1, \dots, m_1 + m_2 \\ a_{i2}x_2 + \cdots + a_{in}x_n &\geq b_i, \quad i = m_1 + m_2 + 1, \dots, m \end{aligned}$$

eliminating variable  $x_1$  gives

$$\begin{aligned} (a_{k2} + a_{i2})x_2 + \cdots + (a_{kn} + a_{in})x_n &\geq (b_i + b_k), \quad i = 1, \dots, m_1, \\ &\quad k = m_1 + 1, \dots, m_1 + m_2 \\ a_{i2}x_2 + \cdots + a_{in}x_n &\geq b_i, \quad i = m_1 + m_2 + 1, \dots, m \end{aligned}$$

The multipliers  $u^l$ ,  $l = 1, \dots, m_1 m_2$  which generate the first set of constraints have a 1 in components  $i$  and  $k$  and all other components are 0. The second set of constraints are not aggregate constraints, they are a copy of the original constraints which had a coefficient of 0 on variable  $x_1$ , therefore the multipliers  $u^l$ ,  $l = m_1 m_2 + 1, \dots, m_1 m_2 + (m - m_1 - m_2)$  generating these constraints have a 1 in component  $i$  for  $i = m_1 + m_2 + 1, \dots, m$  and a zero for all other components.

If one of the aggregate constraints generated is of the form

$$0x_1 + \dots + 0x_n \geq d > 0 \quad (2.15)$$

we conclude that the original system has no solution. This was the case for the  $u$  vectors just given when the right hand side of the third equation in Example 2.9 was changed from -15 to 0. This observation is the basis of the theorems of the alternative developed in Section 5.

3. Each step of Gaussian “projection” eliminates one variable and one constraint. The system (2.5) - (2.7) after projecting out one variable has  $m_1 m_2 + (m - m_1 - m_2)$  constraints. If exactly half the coefficients of  $x_1$  are positive and half the coefficients of  $x_2$  are negative, that is,  $m_1 = m_2 = (1/2)m$ , then the number of constraints in the projected system is  $(m/2)^2$ .
4. In the discussion so far, we assumed that when projecting out variable  $x_i$  it had both positive and negative coefficients. What if this were not the case? That is, what if every coefficient of  $x_i$  is nonnegative (nonpositive). This is easy. Consider the following original system (assume without loss, nonnegative coefficients).

$$\begin{aligned} x_1 + a_{i2}x_2 + \dots + a_{in}x_n &\geq b_i, \quad i = 1, \dots, m_1 \\ a_{i2}x_2 + \dots + a_{in}x_n &\geq b_i, \quad i = m_1 + 1, \dots, m \end{aligned}$$

Projecting out  $x_1$  results in the system

$$a_{i2}x_2 + \dots + a_{in}x_n \geq b_i, \quad i = m_1 + 1, \dots, m.$$

The projection into the variable space without  $x_1$  is found by deleting all the constraints which have a strictly positive coefficient for variable  $x_1$ . The multiplier vectors,  $u^k$ ,  $k = 1, \dots, m - m_1$  which generate the inequalities in the projected space are unit vectors with the one in component  $i$  for  $i = m_1 + 1, \dots, m$ . If all the coefficients on variable  $x_1$  are positive, the original system is

$$x_1 + a_{i2}x_2 + \dots + a_{in}x_n \geq b_i, \quad i = 1, \dots, m,$$

and projecting out  $x_1$  gives  $\mathbb{R}^{n-1}$ . In this case we take the multiplier vector to be  $u = 0$  and the implied inequality is  $0x_2 + \dots + 0x_n \geq 0$ . In this case the polyhedron defined by the inequalities  $Ax \geq b$  is unbounded. Since the  $u$  vector generated at each step of projection is nonnegative, the only way projection can generate a  $u = 0$ , is, if at some point, all the coefficients of a variable are positive (negative).

5. If  $A$  and  $B$  are  $m \times n_1$  and  $m \times n_2$ , respectively, matrices of real numbers and  $b$  an  $m$  vector of real numbers, the statement “*project out the  $x$  variables from the system  $Ax + By \geq b$* ” is always well defined. Since there are  $n_1$   $x$  variables the projection process is finite. By repeated application of Lemmas 2.6 and 2.7 projecting out the  $x$  variables results in a new system with a finite number of linear inequalities which describe the projection into the  $y$  variable subspace. Unfortunately, although finite, projection is not an efficient algorithm in the sense of having a polynomial time bound in the size of the problem input. We leave it as Exercise 2.3 to show that it is possible to construct an  $m$  constraint,  $n$  variable example where the number of constraints generated by projection is

$$(1/2)^{(2^{n+1}-2)}m^{2^n}.$$

Thus, the work of projection is a *doubly exponential* function of the size of the problem input. Weispfenning [447] has developed an elimination method which is singly exponential.

By observations 1, 2, and 4 each step of the projection method generates a set of constraints by aggregating constraints in the original system. Projection provides the multipliers which generate these aggregate constraints. This is formalized as Lemma 2.10.

**Lemma 2.10** *Projection applied to the system  $Ax \geq b$  generates the system  $0 \geq d_k$ ,  $k = 1, \dots, q$ , and for each  $k$ , there is a nonnegative vector  $u^k$ , such that*

$$A^T u^k = 0, \quad b^T u^k = d_k. \quad (2.16)$$

The proof is left as an exercise. This proof should be constructive – show how the  $u^k$  vector is constructed using projection. Refer back to Example 2.9 to see how the multipliers are generated as each variable is eliminated. By observation 4, if at some point in the projection process a variable has all strictly positive coefficients or strictly negative coefficients then the associated

multiplier is  $u^k = 0$  and the polyhedron is unbounded. The following Corollary and Theorem are immediate from repeated application of Proposition 2.8 and Lemma 2.10.

**Corollary 2.11** *Either*

1. *the system  $Ax \geq b$  has a solution and eliminating the  $x$  variables yields the system  $0 \geq d_k, k = 1, \dots, q$  and all  $d_k$  nonpositive, or;*
2. *the system  $Ax \geq b$  has no solution and eliminating the  $x$  variables yields the system  $0 \geq d_k, k = 1, \dots, q$  and at least one  $d_k$  is strictly positive.*

**Theorem 2.12** *If  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ , then projecting out the  $x$  variables in the system  $Ax + By \geq b$  generates the nonnegative vectors  $u^k, k = 1, \dots, q$  such that  $A^\top u^k = 0$  and  $\text{proj}_y(P) = \{y \in \mathbb{R}^{n_2} \mid (u^k)^\top By \geq d_k, k = 1, \dots, q\}$ . If  $q = 1$  and  $u^1 = 0$ , then  $\text{proj}_y(P) = \mathbb{R}^{n_2}$ .*

When using projection to actually find a solution to a system of inequalities one proceeds as in Gaussian elimination. Given a solution to the system with one variable, work recursively backwards to find a solution to the original system.

**Example 2.13** *In Example 2.9 projecting out  $x_1$  and  $x_2$  gave  $1 \leq x_3 \leq 5$ . Take  $\bar{x}_3 = 5$ . Then the system in  $x_2$  and  $x_3$  becomes:*

$$\begin{array}{rcl} x_2 & +(0)5 & \geq & 7 & (1/3)(E1) + (E2) \\ -x_2 & +(3/2)5 & \geq & -11/2 & (1/6)(E1) + (1/2)(E4) \\ -x_2 & -5 & \geq & -12 & (E2) + (E3) \\ -x_2 & +(1/2)5 & \geq & -30/4 & (1/4)(E3) + (1/4)(E4) \end{array}$$

which is equivalent to  $x_2 = 7$ . For  $\bar{x}_2 = 7, \bar{x}_3 = 5$ , the only solution to the original system is  $\bar{x}_1 = 8$ .

## 2.4 APPLICATIONS OF PROJECTION

### 2.4.1 Weyl's Theorem

Later in Chapter 3 (see also Appendix A) we prove the fundamental result that a polyhedral cone is finitely generated. Here we prove the converse of that theorem using projection.

**Theorem 2.14 ( Weyl)** *If the cone  $C$  is generated from a finite set of  $n$  dimensional vectors, then it is a polyhedral cone in  $\mathbb{R}^n$ .*

**Proof:** By hypothesis  $C = \text{cone}\{x^1, \dots, x^q\}$ , that is,

$$C = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q z_i x^i, z_i \geq 0, i = 1, \dots, q\}.$$

Project out the  $z_i$  variables. By Theorem 2.12 this results in either  $\mathbb{R}^n$  which is vacuously a polyhedral cone, or a finite system of linear inequalities involving only the  $x$  variables. Since each inequality in the original system has a zero right hand side, each time a variable is eliminated the resulting system has an all zero right hand side. Therefore, projecting out all the  $z$  variables gives a system of inequalities with zero right hand sides, i.e. a polyhedral cone.  $\square$

This result also extends to polyhedrons. If the set  $P$ , is finitely generated by

$$P = \{x \mid x = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i, \sum_{i=1}^q z_i = 1, z_i \geq 0, i = 1, \dots, r\},$$

then  $P$  is a polyhedron.

## 2.4.2 Solving Linear Programs

It is possible to solve linear programs using projection. Consider the linear program  $\min\{c^\top x \mid Ax \geq b\}$ . Add a variable  $z_0$  to the system and use projection to eliminate the  $x$  variables from the system

$$\begin{aligned} z_0 - c^\top x &\geq 0 \\ Ax &\geq b. \end{aligned}$$

In this system the  $z_0$  variable represents the objective function value and the projection of the  $x$  variables into the  $z_0$  subspace gives the possible objective function values. Eliminating the  $x$  variables results in the system  $z_0 \geq d_k$ ,  $k = 1, \dots, q$  in the  $z_0$  variable and perhaps some inequalities of the form  $0 \geq d_k$ ,  $k = q+1, \dots, r$  which do not involve the  $z_0$  variable. The optimal solution value is

$$z_0 = \max\{d_k \mid k = 1, \dots, q\}.$$

The optimal values of the solution variables  $x$  are recovered in the manner described earlier in Example 2.13. How would you recognize an infeasible linear

program? The linear program  $\min\{c^\top x \mid Ax \geq b\}$  is unbounded if there is an  $\bar{x}$  in the recession cone  $\{x \mid Ax \geq 0\}$  such that  $c^\top \bar{x} < 0$ . How would you recognize an unbounded linear program?

**Example 2.15** Consider the following linear program.

$$\begin{aligned} \min \quad & 2x_1 - 3x_2 \\ \text{s.t.} \quad & x_2 \geq 2 \\ & 0.5x_1 - x_2 \geq -8 \\ & -0.5x_1 + x_2 \geq -3 \\ & x_1 - x_2 \geq -6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Putting this in the appropriate form for projection gives

$$\begin{array}{lllll} z_0 & -2x_1 & +3x_2 & \geq & 0 \quad (E0) \\ & & x_2 & \geq & 2 \quad (E1) \\ & 0.5x_1 & -1x_2 & \geq & -8 \quad (E2) \\ & -0.5x_1 & +x_2 & \geq & -3 \quad (E3) \\ & x_1 & -x_2 & \geq & -6 \quad (E4) \\ & x_1 & & \geq & 0 \quad (E5) \\ & x_2 & & \geq & 0 \quad (E6) \end{array}$$

Even though the nonnegativity constraint  $x_2 \geq 0$  is redundant because of the constraint  $x_2 \geq 2$  we carry it along. It provides information in a later section on sensitivity analysis. A constraint  $(a^k)^\top x \geq b_k$  which is part of the system  $Ax \geq b$  is a redundant constraint if and only if removing the constraint from the system does not alter the feasible region of the associated polyhedron. First eliminate variable  $x_1$ . This gives the system

$$\begin{array}{llll} z_0 & -x_2 & \geq & -32 \quad (E0) + 4(E2) \\ z_0 & +x_2 & \geq & -12 \quad (E0) + 2(E4) \\ z_0 & +3x_2 & \geq & 0 \quad (E0) + 2(E5) \\ & x_2 & \geq & -12 \quad 2(E3) + (E4) \\ & 2x_2 & \geq & -6 \quad 2(E3) + (E5) \\ & x_2 & \geq & 2 \quad (E1) \\ & x_2 & \geq & 0 \quad (E6) \\ 0 & \geq & -11 & (E2) + (E3) \end{array}$$

Now eliminate variable  $x_2$ .

$$\begin{aligned}
 z_0 &\geq -22 & (E0) + 2(E2) + (E4) \\
 z_0 &\geq -24 & (E0) + 3(E2) + 0.5(E5) \\
 z_0 &\geq -44 & (E0) + 4(E2) + 2(E3) + (E4) \\
 z_0 &\geq -35 & (E0) + 4(E2) + (E3) + 0.5(E5) \\
 z_0 &\geq -30 & (E0) + (E1) + 4(E2) \\
 z_0 &\geq -32 & (E0) + 4(E2) + (E6) \\
 0 &\geq -11 & (E2) + (E3)
 \end{aligned}$$

Making  $z_0$  as small as possible gives  $z_0 = -22$ . If  $z_0 = -22$  then the equations involving  $x_2$  and  $z_0$  become

$$\begin{aligned}
 -x_2 &\geq -10 & (E0) + 4(E2) \\
 x_2 &\geq 10 & (E0) + 2(E4) \\
 3x_2 &\geq 22 & (E0) + 2(E5) \\
 x_2 &\geq -12 & 2(E3) + (E4) \\
 2x_2 &\geq -6 & 2(E3) + (E5) \\
 x_2 &\geq 2 & (E1) \\
 x_2 &\geq 0 & (E6) \\
 0 &\geq -11 & (E2) + (E3)
 \end{aligned}$$

A feasible solution (and also unique) to this system is  $x_2 = 10$ . Repeating this process for the system with  $z_0$  fixed at  $-22$  and  $x_2$  fixed at  $10$  gives the unique value  $x_1 = 4$ . One can also find the optimal  $x_1$  and  $x_2$  by observing that for the binding constraint  $z_0 = -22$  on the objective function value, constraints (E2) and (E4) are aggregated to give the value of  $-22$ . If these inequalities are written as equalities they can be solved simultaneously to obtain the optimal  $x_1 = 4$ ,  $x_2 = 10$  solution.

## 2.5 THEOREMS OF THE ALTERNATIVE

The results developed in this section are usually summarized as *theorems of the alternative*. These theorems boil down to nothing more than saying either the system  $Ax \geq b$  has a solution or it does not. (As if we didn't know that!) What distinguishes these theorems is how they characterize a system which has no solution. A system is characterized as not solvable when it is possible to generate an aggregate constraint that clearly has no solution.

Lemma 2.10 and Corollary 2.11 are often combined into the famous result known as *Farkas' lemma* which says either the system  $Ax \geq b$  has a solution

or it does not. Farkas' lemma is a fundamental result in the theory of linear inequalities.

**Lemma 2.16 ( Farkas)** *Either the system*

$$Ax \geq b$$

*has a solution, or the system*

$$A^T u = 0, \quad b^T u > 0, \quad u \geq 0$$

*has a solution, but not both.*

**Proof:** Clearly both systems cannot have a solution  $(\bar{x}, \bar{u})$  because  $\bar{u} \geq 0$  and  $A\bar{x} \geq b$  implies  $\bar{u}^T A\bar{x} \geq \bar{u}^T b > 0$  which contradicts  $A^T \bar{u} = 0$ . To see that at least one system has a solution simply observe that either  $Ax \geq b$  has a solution or it does not. If  $Ax \geq b$  has a solution we are done. Assume  $Ax \geq b$  has no solution. Then by Corollary 2.11, applying projection to the system  $Ax \geq b$  gives an inequality  $0 \geq d_k > 0$ . Then by Lemma 2.10 there exists a nonnegative  $u^k$  such that  $A^T u^k = 0$  and  $b^T u^k > 0$ .  $\square$

Kuhn [1956] calls the system  $Ax \geq b$  *inconsistent* if there is a nonnegative vector  $u \geq 0$  such that  $A^T u = 0$  and  $b^T u > 0$ . We have shown that the system  $Ax \geq b$  is inconsistent if and only if it does not have a solution. The following is a simple corollary to Farkas' lemma. It is often referred to as the fundamental theorem of linear algebra or the first Fredholm theorem.

**Corollary 2.17 ( Fredholm Alternative)** *Either the system*

$$Ax = b$$

*has a solution, or the system*

$$A^T u = 0, \quad b^T u \neq 0$$

*has a solution, but not both.*

**Proof:** Write the system  $Ax = b$  as  $Ax \geq b$ , and  $-Ax \geq -b$  and apply Farkas' lemma.  $\square$

**Corollary 2.18** Either the system

$$Ax \geq b, \quad x \geq 0$$

has a solution, or the system

$$A^T u \leq 0, \quad u \geq 0, \quad b^T u > 0$$

has a solution, but not both.

**Corollary 2.19** Either the system

$$Ax = b, \quad x \geq 0$$

has a solution, or the system

$$A^T u \leq 0, \quad b^T u > 0$$

has a solution, but not both.

There is a nice geometric interpretation to Corollary 2.19. Generate the cone  $C \subseteq \mathbb{R}^m$  by taking the conic hull of  $\{a^1, \dots, a^n\}$ . Corollary 2.19 states that the vector  $b$  is either in this cone (i.e.  $b$  is a conic combination of  $a^1, \dots, a^n$ ) or, if it is not in the cone  $C$ , we can find a vector  $u$  which makes an angle of strictly less than 90 degrees with  $b$ , ( $b^T u > 0$ ) but is 90 degrees or more away from each of the vectors  $\{a^1, \dots, a^n\}$ .

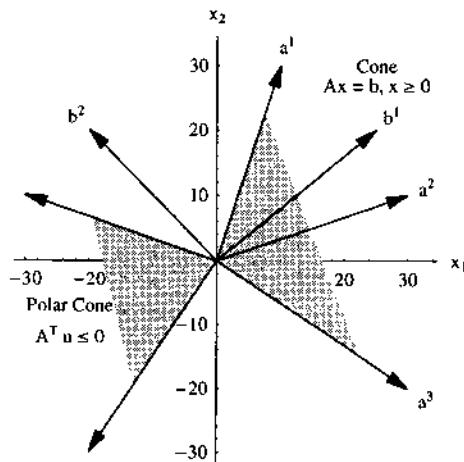
In Figure 2.2, the vector  $b^1$  is in the cone defined by  $Ax = b$ ,  $x \geq 0$  and  $b^1$  makes at least a 90 degree angle with every vector  $u$  which makes at least a 90 degree angle with all of the generators of the cone, i.e  $A^T u \leq 0$ . The vector  $b^2$  is not in the cone and is within 90 degrees of every vector that makes at least a 90 degree angle with all of the cone generators.

Corollary 2.19 can be used to characterize the polar cone of polyhedral cones. If  $K$  is a cone, the *polar cone*  $K^*$  of  $K$ , is defined by

$$K^* = \{b \mid b^T u \leq 0, \text{ for all } u \in K\}.$$

If  $K = \{u \mid A^T u \leq 0\}$  and  $K' = \{b \mid b = Ax, x \geq 0\}$  it is obvious that  $K' \subseteq K^*$  since  $b \in K'$  and  $u \in K$  implies  $b^T u = (Ax)^T u = x^T (A^T u) \leq 0$  for any  $x \geq 0$ . By Corollary 2.19 it also follows that  $K^* \subseteq K'$  and we have the following corollary characterizing the polyhedral cone  $K$  and its polar cone  $K^*$ .

Figure 2.2 The Polar Cone



**Corollary 2.20** If  $K = \{u \mid A^T u \leq 0\}$ , then  $K^* = \{b \mid b = Ax, x \geq 0\}$ .

**Corollary 2.21** If there is no solution to the system

$$A_1 x > b^1, \quad A_2 x \geq b^2$$

then there are nonnegative  $u^1, u^2$  such that

$$A_1^T u^1 + A_2^T u^2 = 0, \quad (b^1)^T u^1 + (b^2)^T u^2 \geq 0, \quad u^1 \neq 0 \quad (2.17)$$

has a solution, or

$$A_1^T u^1 + A_2^T u^2 = 0, \quad (b^1)^T u^1 + (b^2)^T u^2 > 0, \quad (2.18)$$

has a solution. Conversely, if there is no solution to either (2.17) or (2.18), there is a solution to  $A_1 x > b^1, A_2 x \geq b^2$ .

In Farkas' lemma and all of its variants, the key idea is to take a system of inequalities and equalities and generate an aggregate constraint from that system. If the original system has no solution, the aggregate constraint must obviously have no solution (for example  $0^T x \geq d > 0$  or  $0^T x = d \neq 0$ ). The "alternative" system corresponds to constraints on the aggregate constraint coefficients (e.g.  $A^T u = 0$ ) and constraints which indicate that the aggregate

constraint is a valid aggregation on the original system (e.g.  $u \geq 0$ ). Then, if the original system has a solution it must be impossible to generate such a constraint. If it does not have a solution then it is possible to generate such an aggregate constraint.

Given polyhedron  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ , let  $(u^k)^T By \geq d_k$ ,  $k = 1, \dots, q$  denote the system that results from projecting out the  $x$  variables. From Theorem 2.12 this system defines the projection of  $P$  into the subspace of  $y$  variables, i.e.

$$\text{proj}_y(P) = \{y \in \mathbb{R}^{n_2} \mid (u^k)^T By \geq d_k, k = 1, \dots, q\}.$$

Another way to characterize  $\text{proj}_y(P)$  is through a projection cone. See, for example, Balas and Pulleyblank [29]. Define the *projection cone* of  $P$  with respect to the  $x$  variables by  $C_x(P) = \{u \in \mathbb{R}^m \mid A^T u = 0, u \geq 0\}$ .

**Proposition 2.22** *If  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ , then*

$$\text{proj}_y(P) = \{y \in \mathbb{R}^{n_2} \mid (u^i)^T By \geq (u^i)^T b, i = 1, \dots, r\}$$

where the vectors  $u^1, \dots, u^r$  denote the extreme rays of the projection cone  $C_x(P) = \{u \in \mathbb{R}^m \mid A^T u = 0, u \geq 0\}$ .

**Proof:** Let  $\text{proj}'_y(P) = \{y \in \mathbb{R}^{n_2} \mid (u^i)^T By \geq (u^i)^T b, i = 1, \dots, r\}$  and show that  $\text{proj}'_y(P) = \text{proj}_y(P)$ . It is obvious that  $\text{proj}_y(P) \subseteq \text{proj}'_y(P)$  since any  $(u^i)^T By \geq (u^i)^T b$  is an aggregate constraint of the system  $Ax + By \geq b$  for any extreme ray  $u^i$  of the projection cone. Show  $\text{proj}'_y(P) \subseteq \text{proj}_y(P)$ . Assume not, i.e. there exists  $\bar{y} \in \text{proj}'_y(P)$  but  $\bar{y} \notin \text{proj}_y(P)$ . By definition of projection,  $\bar{y} \notin \text{proj}_y(P)$  implies there is no solution to the system  $Ax \geq b - B\bar{y}$ . Then by Farkas' lemma there exists a vector  $\bar{u} \geq 0$  such that  $A^T \bar{u} = 0$  and  $(b - B\bar{y})^T \bar{u} > 0$ . Therefore,  $\bar{u} \in C_x(P)$ . Then  $\bar{u}$  is a conic combination of the  $u^i, i = 1, \dots, r$ . (See Theorem 3.3 or A.15) But by definition of  $\text{proj}'_y(P)$ ,  $(b - B\bar{y})^T u^i \leq 0$  for  $i = 1, \dots, r$  contradicting  $(b - B\bar{y})^T \bar{u} > 0$ .  $\square$

By Theorem 2.12, the system of inequalities  $(u^k)^T By \geq d_k, k = 1, \dots, q$  that results from projecting out the  $x$  variables from the system  $Ax + By \geq b$  defines  $\text{proj}_y(P)$ . Since the rays of the projection cone  $C_x(P)$  also generate  $\text{proj}_y(P)$ , will the set of multipliers generated using projection contain all the extreme rays of  $C_x(P)$ ?

**Proposition 2.23** *If  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ , and the vectors  $u^i, i = 1, \dots, q$  are the multipliers generated by projection, then the extreme*

*rays of the projection cone  $C_x(P) = \{u \in \mathbb{R}^m \mid A^\top u = 0, u \geq 0\}$  are contained in this set of multipliers.*

**Proof:** Proof by induction on the value of  $n_1$ . Assume  $n_1 = 1$ . Then without loss, the matrix  $A$  is a single column,  $a^1$ , with elements  $0, \pm 1$ . The projection cone is  $\{u \mid (a^1)^\top u = 0, u \geq 0\}$ . The extreme rays of this cone are very easy to characterize. The set of extreme rays consists of all  $u$  vectors where all components are zero except for two components  $i, k$  which are  $\pm 1$  and  $a_{i1} = 1, a_{k1} = -1$ . These extreme rays correspond exactly to the multipliers generated from projection.

Now assume the result is true for  $n_1 - 1$ . Show that the result is valid for  $n_1$ . Let  $\bar{u}$  be an extreme ray of  $C_x(P)$  and let  $A'$  denote the matrix obtained from  $A$  by deleting column  $n_1$ . If the multipliers  $w^1, \dots, w^t$  are generated by eliminating the first  $n_1 - 1$  variables of  $A$ , then by the induction hypothesis these multipliers contain the extreme rays of the projection cone  $\{u \mid (A')^\top u = 0, u \geq 0\}$ . But  $C_x(P) \subseteq \{u \mid (A')^\top u = 0, u \geq 0\}$ . So  $\bar{u}$  is a conic combination of  $w^1, \dots, w^t$ . Then there exists nonnegative  $\bar{\lambda}_i$  such that  $\bar{u} = \sum_{i=1}^t \bar{\lambda}_i w^i$ . Without loss,  $(a^{n_1})^\top w^i = 0, \pm 1, i = 1, \dots, t$ . Let  $I^+ = \{i \mid (a^{n_1})^\top w^i = +1, i = 1, \dots, t\}$  and  $I^- = \{i \mid (a^{n_1})^\top w^i = -1, i = 1, \dots, t\}$ . Since  $(a^{n_1})^\top \bar{u} = 0$ ,  $\sum_{i=1}^t \bar{\lambda}_i (a^{n_1})^\top w^i = 0$ . It follows that there exist positive  $\bar{\theta}_{ij}$  such that  $i \in I^+, j \in I^-$  and  $\sum_{i \in I^+} \sum_{j \in I^-} \bar{\theta}_{ij} (w^i + w^j) = \sum_{i=1}^t \bar{\lambda}_i w^i = \bar{u}$ . Now eliminate variable  $n_1$  in the  $A$  matrix and let  $u^l, l = 1, \dots, q$  be the corresponding multipliers. By construction for every  $i \in I^+$  and  $j \in I^-$  there is an  $u^l = w^i + w^j$ . Therefore,  $\bar{u}$  is a conic combination of the  $u^l, l = 1, \dots, q$ .  $\square$

The system of inequalities  $(u^k)^\top By \geq d_k, k = 1, \dots, q$  that results from projecting out the  $x$  variables from the system  $Ax + By \geq b$ , in general, will contain many inequalities redundant to the description of  $\text{proj}_y(P)$ . Proposition 2.22 provides a sufficient condition for recognizing many of these redundant inequalities. By Proposition 2.22 the inequalities obtained from the extreme rays give a complete characterization of the projected polytope. Therefore, any inequality  $(u^k)^\top By \geq d_k$  not generated by an extreme ray is redundant. This is formalized in Proposition 2.24 which combines a result of Kohler [269] and Motzkin [347].

**Proposition 2.24** *If matrix  $A$  has rank  $t$  and  $(u^k)^\top By \geq d_k, k = 1, \dots, q$  is the system of inequalities that results from projecting out the  $x$  variables from the polyhedron*

$$P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\},$$

then any inequality  $(u^k)^\top By \geq d_k$  which is not redundant is the aggregation of  $t+1$  inequalities from the system  $Ax + By \geq b$  and the rank of the rows of  $A$  aggregated to form  $u^k$  is  $t$ .

**Proof:** Consider the projection cone  $C_x(P) = \{u \in \mathbb{R}^m \mid A^\top u = 0, u \geq 0\}$ . Let  $\bar{u}$  be an extreme ray of  $C_x(P)$ . If  $J = \{i \mid \bar{u}_i > 0\}$  then  $\bar{u}$  is a nonzero solution to the system

$$A^\top u = 0, \quad u_i = 0, \quad i \notin J. \quad (2.19)$$

Because  $\bar{u}$  is an extreme ray of  $C_x(P)$ , the coefficient matrix of system (2.19) has rank  $m-1$  which implies that the rows of  $A$  indexed by  $J$  must have rank  $(|J|-1)$ . Since  $A$  has rank  $t$ , it follows that  $|J| \leq t+1$ . If  $|J| < t+1$  for extreme ray  $\bar{u}$ , it is always possible to define an index set  $\hat{J}$  such that  $J \subset \hat{J}$  where the rows of  $A$  indexed by  $\hat{J}$  have rank  $t$ . By Proposition 2.22 any inequality  $(u^k)^\top By \geq d_k$  which does not correspond to an extreme ray is redundant and the proposition is proved.  $\square$

An even simpler test for redundant constraints is given by Corollary 2.25.

**Corollary 2.25 (Kohler)** *If  $r$  variables have been eliminated using projection, then any resulting constraint which is the aggregation of more than  $r+1$  of the original constraints is redundant.*

Although Proposition 2.24 and Corollary 2.25 provide a sufficient condition for screening redundant inequalities, they do not provide necessary conditions. Unfortunately, even if the row vector  $\bar{u}$  is an extreme ray of the projection cone  $C_x(P)$ , the inequality  $\bar{u}^\top Ay \geq \bar{u}^\top b$  may be redundant and therefore not needed in the description of  $\text{proj}_y(P)$ . Balas [24] shows how to transform the inequalities defining  $P$  so that the extreme rays of the resulting projection cone define the nonredundant inequalities of the projected polyhedron. The results of this section are often combined into Theorem 2.26 which strengthens Corollary 2.19.

**Theorem 2.26** *If  $\{a^1, a^2, \dots, a^n, b\}$  is a set of vectors in  $\mathbb{R}^m$ , with rank  $t$ , then either*

1. *the system  $(a^i)^\top u \leq 0, i = 1, \dots, n \quad b^\top u > 0$  has a solution  $\bar{u}$ , and the rank of  $\{a^i \mid (a^i)^\top \bar{u} = 0, i \in \{1, \dots, n\}\}$  is  $t-1$ , or*

2.  $b$  is a nonnegative (conic) combination of linearly independent vectors from  $\{a^1, \dots, a^n\}$ ,

but not both.

**Proof:** It is easy to show that both condition 1 and 2 cannot hold simultaneously so we will show that either condition 1, or condition 2 must hold. Either there is a solution  $\bar{u}$  such that  $b^\top \bar{u} > 0$  and  $(a^i)^\top \bar{u} \leq 0, i = 1, \dots, n$  or there is not. Assume such a  $\bar{u}$  exists. That is, the polyhedral cone

$$C = \{u \mid b^\top u \geq 0, (a^i)^\top u \leq 0, i = 1, \dots, n\}$$

has a nonzero element  $\bar{u}$  such that  $b^\top \bar{u} > 0$ . Since  $\bar{u}$  is not in the lineality space of  $C$  it can be written as a conic combination of  $u^i, i = 1, \dots, r$  where each  $u^i$  is an element of a proper minimal face of  $C$ . See Theorem A.15 in Appendix A. Without loss, we can assume  $b^\top u^1 > 0$ . Since the rank of the vectors  $a^1, \dots, a^n, b$  is  $t$ , it follows from applying Proposition A.2 and Lemma A.13 in Appendix A to the minimal face containing  $(u^1)^\top$  that the rank of the vectors  $\{a^i \mid (a^i)^\top u^1 = 0, i \in \{1, \dots, n\}\}$  is  $t - 1$ . Thus, condition 1 is satisfied.

Assume there is no solution  $\bar{u}$  such that  $b^\top \bar{u} > 0$  and  $(a^i)^\top \bar{u} \leq 0, i = 1, \dots, n$ . Then there is an optimal solution to the linear program  $\max \{b^\top u \mid (a^i)^\top u \leq 0\}$ . If the  $u$  variables are eliminated from the system

$$\begin{aligned} z_0 - b^\top u &\leq 0, \\ (a^i)^\top u &\leq 0, \quad i = 1, \dots, n, \end{aligned}$$

there must be at least one inequality of the form  $z_0 \leq 0$  in the resulting system otherwise the projection into the  $z_0$  variable subspace would be  $\mathbb{R}^1$  and the linear program unbounded. Then by Proposition 2.24 there exists an index set  $\hat{J} \subseteq \{1, \dots, n\}$  of cardinality  $t$  such that

$$-b\bar{x}_0 + \sum_{i \in \hat{J}} a^i \bar{x}_i = 0$$

where  $\bar{x}_0 > 0$  (if  $\bar{x}_0 = 0$  then the aggregate constraint would not contain variable  $z_0$ ),  $\bar{x}_i \geq 0$  for all  $i \in \hat{J}$  and the rank of the set of vectors consisting of  $b$  and the  $a^i$  indexed by  $\hat{J}$  is  $t$ . Then  $b$  is a conic, and therefore linear, combination of vectors  $a^i, i \in \hat{J}$  and it follows that the vectors  $a^i, i \in \hat{J}$  have rank  $t$  and therefore are linearly independent.  $\square$

Finally, following Bachem and Kern [18] (Chapter 4), we see that the results of this section are easily extended from polyhedra to linear subspaces.

**Corollary 2.27** *If  $L$  is a linear subspace of  $\mathbb{R}^m$  and  $i \in \{1, \dots, m\}$ , then either*

*there is  $z \in L$ ,  $z \geq 0$ ,  $z_i > 0$ ,*

*or*

*there is  $u \in L^\perp$ ,  $u \geq 0$ ,  $u_i > 0$ ,*

*but not both.*

**Proof:** If  $L$  is a subspace of  $\mathbb{R}^m$  then there exists an  $m \times n$  matrix  $A$  such that  $L = \{z \in \mathbb{R}^m \mid \text{there is } x, z = Ax\}$ . Define  $A_1$  to be the submatrix of  $A$  consisting of only row  $i$  and  $A_2$  to be the submatrix of  $A$  consisting of the rows  $\{1, \dots, m\} \setminus \{i\}$ . Observe that  $L^\perp = \{u \in \mathbb{R}^m \mid A^\top u = 0\}$ . The result now follows from Corollary 2.21.  $\square$

## 2.6 DUALITY THEORY

Duality theory is one of the most important aspects of linear programming. We motivate duality theory by revisiting Example 2.15. This linear program had the form  $\min\{c^\top x \mid Ax \geq b\}$  and was optimized by eliminating the  $x$  variables from the system,

$$\begin{aligned} z_0 - c^\top x &\geq 0 \\ Ax &\geq b \end{aligned}$$

and finding the minimum feasible value for  $z_0$  in the resulting system

$$\begin{aligned} z_0 &\geq d_k, \quad k = 1, \dots, q \\ 0 &\geq d_k, \quad k = q+1, \dots, r. \end{aligned}$$

The minimum value is  $z_0 = \max\{d_k \mid k = 1, \dots, q\}$  since we must *maximize* over the  $d_k$  in order to have a feasible value for  $z_0$ . Then every nonnegative vector of multipliers  $u^k, k = 1, \dots, q$  where  $A^\top u^k = c$  and  $d_k = b^\top u^k$  provides a lower bound on the optimal linear programming value. This idea is known as *weak duality*.

**Lemma 2.28 ( Weak Duality )** *If  $\bar{x}$  is a solution to the system  $Ax \geq b$  and  $\bar{u} \geq 0$  is a solution to  $A^\top u = c$ , then  $c^\top \bar{x} \geq b^\top \bar{u}$ .*

**Proof:** Since  $\bar{u} \geq 0$ ,  $\bar{u}^\top Ax \geq \bar{u}^\top b$  is an aggregate constraint of the system  $Ax \geq b$ . If  $A\bar{x} \geq b$ , then  $\bar{u}^\top A\bar{x} \geq \bar{u}^\top b$ . By hypothesis  $c = A^\top \bar{u}$ , so  $c^\top \bar{x} = \bar{u}^\top A\bar{x} \geq \bar{u}^\top b$   $\square$

Since the minimum value of the linear program is given by  $z_0 = \max\{d_k \mid k = 1, \dots, q\}$  it is necessary to find a nonnegative  $u$  such that  $A^\top u = c$  and  $b^\top u$  is as large as possible. Finding the largest value of  $b^\top u$  is also a linear program:

$$\max \{b^\top u \mid A^\top u = c, u \geq 0\}.$$

In general, the linear program  $\min\{c^\top x \mid Ax \geq b\}$  under consideration is called the *primal linear program* and the linear program  $\max\{b^\top u \mid A^\top u = c, u \geq 0\}$  which generates the multipliers for the lower bounds is called the *dual linear program*. The  $x$  variables are the primal variables and the  $u$  variables the dual variables or dual multipliers. When the primal has an optimal solution there is also an optimal solution to the dual linear program and the objective function values are equal.

**Theorem 2.29 ( Strong Duality )** *If there is an optimal solution to the primal linear program  $\min\{c^\top x \mid Ax \geq b\}$ , then there is an optimal solution to the dual linear program  $\max\{b^\top u \mid A^\top u = c, u \geq 0\}$  and the optimal objective function values are equal.*

**Proof:** Apply projection to  $\min\{c^\top x \mid Ax \geq b\}$ . Add variable  $z_0$  and define the system

$$z_0 - c^\top x \geq 0 \tag{2.20}$$

$$Ax \geq b. \tag{2.21}$$

Eliminating the  $x$  variables gives the system (after possible scaling of the coefficients on the  $z_0$  variable)

$$z_0 \geq d_k, k = 1, \dots, q, \quad 0 \geq d_k, k = q + 1, \dots, r. \tag{2.22}$$

Since the primal is feasible,  $d_k \leq 0$  for all  $k = q + 1, \dots, r$ . Since the primal has an optimal solution,  $q \geq 1$  in (2.22), otherwise the projection of (2.20) - (2.21) into the  $z_0$  variable subspace is  $\mathbb{R}^1$  implying that the linear program is unbounded. Then the optimal solution value of the primal problem is  $\bar{z}_0 = \max\{d_k \mid k = 1, \dots, q\}$ . Assume this maximum value is obtained for  $d_1$ . By Lemma 2.10 there exists a nonnegative  $u^1$  such that  $b^\top u^1 = d_1$  and  $A^\top u^1 - c = 0$ . Then  $u^1$  is a feasible dual solution such that  $b^\top u^1 = \bar{z}_0$ . Then by weak

duality  $u^1$  is an optimal dual solution and the optimal primal-dual objective function values are equal.  $\square$

The following corollary is an immediate consequence of weak duality.

**Corollary 2.30** *If the primal linear program  $\min\{c^T x \mid Ax \geq b\}$  is unbounded then the dual linear program  $\max\{b^T u \mid A^T u = c, u \geq 0\}$  is infeasible.*

There are primal-dual pairs other than the pair of Theorem 2.29. For example, if the primal problem is written as

$$\begin{array}{ll} \min & c^T x \\ & Ax \geq b \\ & x \geq 0 \end{array}$$

then the symmetric dual problem is

$$\begin{array}{ll} \max & b^T u \\ & A^T u \leq c \\ & u \geq 0 \end{array}$$

To see why this is the appropriate dual problem let  $u$  be a set of nonnegative multipliers for the  $Ax \geq b$  constraints and  $w$  a set of multipliers for the  $x \geq 0$  constraints. We want an aggregate constraint which maximizes  $b^T u + 0^T w = b^T u$  subject to  $A^T u + Iw = c$ . (Where  $I$  is an  $n \times n$  identity matrix - assume  $x$  is an  $n$ -vector.) Since  $w$  is required to be nonnegative this is equivalent to finding a nonnegative solution to  $A^T u \leq c$ .

If the primal problem is

$$\begin{array}{ll} \min & c^T x \\ & Ax = b \\ & x \geq 0 \end{array}$$

what is the appropriate dual problem?

Thinking about how to solve the primal linear program using projection will always give the dual problem. If the objective function is  $\min c^T x$  then add the constraint  $z_0 \geq c^T x$  to the primal constraints and write all the primal constraints in either  $\geq$  or  $=$  to form. If the  $x$  variables are eliminated using

**Table 2.1** Primal-Dual Relationships

Primal	Dual
inequality primal constraint	nonnegative dual variable
equality primal constraint	unrestricted dual variable
nonnegative primal variable	inequality dual constraint
unrestricted primal variable	equality dual constraint
min primal objective function	max dual objective function

projection, what do the resulting aggregate inequalities look like? Well, the coefficients on the  $x$  variables must be zero since the  $x$  variables are “eliminated.” *The dual constraints are simply the constraints that require the coefficients of the  $x$  variables in the aggregate constraints to be 0.* The dual multipliers on the inequality constraints must be nonnegative and the multipliers on equality constraints are unconstrained. This logic provides the constraints on the dual variables. Because the primal constraint is  $z_0 \geq c^\top x$  we have to find the largest right hand side for a feasible value of  $z_0$ . Therefore, the dual objective function is to maximize the right hand side of the aggregate constraints. Similarly if your primal problem is a maximization problem. This method never fails! Basic primal-dual relationships are summarized in Table 2.1.

An interesting feature of projection is that it generates all of the extreme points dual polyhedron and extreme rays of the recession cone of the dual polyhedron.

**Proposition 2.31** *If the linear program  $\{c^\top x \mid Ax \geq b\}$  has an optimal solution, then applying projection to the system  $z_0 - c^\top x \geq 0$ ,  $Ax \geq b$  gives all of the extreme points of the dual polyhedron  $\{u \mid A^\top u = c, u \geq 0\}$  and all of the extreme rays of the dual recession cone  $\{u \mid A^\top u = 0, u \geq 0\}$ .*

**Proof:** Since the linear program has an optimal solution applying projection to the system

$$z_0 - c^\top x \geq 0, \quad Ax \geq b$$

gives

$$z_0 \geq d_k, \quad k = 1, \dots, q, \quad 0 \geq d_k, \quad k = q+1, \dots, r.$$

Apply Proposition 2.23 and observe that projection generates the extreme rays of the cone  $\{(u, u_0) \mid A^\top u - cu_0 = 0, (u, u_0) \geq 0\}$ . Without loss, take  $u_0 = 1$  in any extreme ray with  $u_0 > 0$ . Then,  $u^i, i = 1, \dots, q$  contains the extreme points of the dual polyhedron and  $u^i, i = q+1, \dots, r$  contains the extreme rays of the dual recession cone.  $\square$

Another interesting primal-dual relation is the result of Clark [94].

**Proposition 2.32 (Clark)** *If either the primal polyhedron  $\{x \mid Ax \geq b, x \geq 0\}$  or dual polyhedron  $\{u \mid A^T u \leq c, u \geq 0\}$  is feasible, then at least one of the two polyhedrons is unbounded.*

**Proof:** Apply projection to the system  $Ax \geq b, x \geq 0$ . Let  $u$  be the multipliers for  $Ax \geq b$  and  $w$  the multipliers for  $x \geq 0$ . By Theorem 2.12, projection either 1) generates a nonzero  $(\bar{u}, \bar{w})$  such that  $A^T \bar{u} + \bar{w} = 0$ , or 2) concludes that the primal polyhedron is unbounded. (See also observation 4 in Section 2.3.) Case 1) implies a nonzero element in the recession cone of the dual polyhedron. If the dual polyhedron is not empty then it is unbounded. If the dual polyhedron is empty then apply projection to the dual system and generate a nonzero element of the primal recession cone. In this case the primal must be feasible and therefore the primal polyhedron is unbounded. In case 2) the primal polyhedron is unbounded.  $\square$

## 2.7 COMPLEMENTARY SLACKNESS

The linear program,  $\min \{c^T x \mid Ax \geq b\}$ , is solved by eliminating the  $x$  variables from the system  $z_0 \geq c^T x, Ax \geq b$ , generating the aggregate constraints  $c^T x = ((u^k)^T A)x \geq d_k$  and finding  $\bar{z}_0 = \max\{d_k \mid k = 1, \dots, q\}$ . If  $\bar{x}$  is an optimal solution to the primal linear program then

$$c^T \bar{x} = ((u^{\hat{k}})^T A) \bar{x} = (u^{\hat{k}})^T b = d_{\hat{k}}$$

for  $\hat{k} = \operatorname{argmax}\{d_k \mid k = 1, \dots, q\}$ . A simple consequence of this, shown below, is that every optimal dual solution must be complementary in the sense that if there is positive slack on a primal constraint the associated dual variable must be zero. Similarly for the dual problem. This fundamental result of linear programming, indeed for even more general mathematical programs, is *complementary slackness*.

**Theorem 2.33** *If  $\bar{x}$  is a feasible solution to the primal problem  $\min \{c^T x \mid Ax \geq b\}$  and  $\bar{u}$  is a feasible solution to the dual problem  $\max \{b^T u \mid A^T u = c, u \geq 0\}$ , then  $\bar{x}, \bar{u}$  are primal-dual optimal if and only if*

$$(b - A\bar{x})^T \bar{u} = 0. \quad \text{primal complementary slackness}$$

**Proof:** Assume  $(\bar{x}, \bar{u})$  is primal-dual optimal. Using constraints  $Ax \geq b$  form the aggregate constraint

$$c^T \bar{x} = (\bar{u}^T A) \bar{x} \geq \bar{u}^T b. \quad (2.23)$$

If  $A_{i\bullet} \bar{x} > b_i$  (where  $A_{i\bullet}$  denotes row  $i$  of matrix  $A$ ) and  $\bar{u}_i > 0$  then (2.23) holds as a strict inequality for  $\bar{x}$ . This is impossible because strong duality implies  $c^T \bar{x} = \bar{u}^T b$ . Therefore, if  $\bar{u}_i > 0$  then  $(b_i - A_{i\bullet} \bar{x}) = 0$  which is primal complementary slackness.

Now assume  $(\bar{x}, \bar{u})$  is primal-dual feasible and the complementary slackness conditions hold. Then

$$(b - A\bar{x})^T \bar{u} = 0$$

that is,

$$b^T \bar{u} = (A\bar{x})^T \bar{u}.$$

Because  $\bar{u}$  is dual feasible  $A^T \bar{u} = c$ . Thus  $b^T \bar{u} = c^T \bar{x}$ . If the primal and dual objective function values are equal they are optimal by weak duality.  $\square$

An immediate corollary is :

**Corollary 2.34** *If*

$$A\bar{x} \geq b \quad (2.24)$$

$$A^T \bar{u} = c \quad \bar{u} \geq 0 \quad (2.25)$$

$$(b - A\bar{x})^T \bar{u} = 0 \quad (2.26)$$

*then  $\bar{x}$  is optimal to the primal problem  $\min \{c^T x \mid Ax \geq b\}$  and  $\bar{u}$  is optimal to the dual problem  $\max \{b^T u \mid A^T u = c, u \geq 0\}$ .*

These complementary slackness conditions are also called *optimality conditions* since they are both sufficient and necessary for a primal-dual pair to be optimal. The complementary slackness conditions state that if there is positive slack on a primal constraint in an optimal solution then the corresponding optimal dual variable has value zero. However, it is possible to have an optimal primal-dual pair with zero slack on a constraint and associated dual variable value of zero. In the following proposition due to Goldman and Tucker [190] we show that it is always possible to find an optimal primal-dual solution where zero slack implies a positive dual variable value.

**Theorem 2.35 ( Strict Complementary Slackness)** *If there is an optimal solution to the linear program  $\min \{c^\top x \mid (a^i)^\top x \geq b_i, i = 1, \dots, m\}$ , then there is an optimal primal-dual pair  $(\bar{x}, \bar{u})$  such that*

$$(b_i - (a^i)^\top \bar{x}) < 0 \Rightarrow \bar{u}_i = 0 \text{ and } (b_i - (a^i)^\top \bar{x}) = 0 \Rightarrow \bar{u}_i > 0, i = 1, \dots, m. \quad (2.27)$$

**Proof:** It suffices to show that (2.27) holds for the arbitrary index 1 since taking a convex combination of optimal primal-dual solutions which satisfy the strict complementary conditions for each constraint gives an optimal primal-dual pair which satisfies strict complementary slackness for every constraint. We show either there is an optimal primal-dual solution  $\bar{x}, \bar{u}$  such that  $\bar{u}_1 > 0$  and  $(b_1 - (a^1)^\top \bar{x}) = 0$  or  $(b_1 - (a^1)^\top \bar{x}) < 0$  and  $\bar{u}_1 = 0$ . Let  $\bar{z}_0$  be the optimal primal-dual solution value and define the system:

$$\bar{z}_0 \geq c^\top x, \quad (a^1)^\top x > b_1, \quad (a^i)^\top x \geq b_i, \quad i = 2, \dots, m. \quad (2.28)$$

If  $\bar{x}$  is a feasible solution to (2.28) then  $\bar{x}$  is also an optimal primal solution and  $\bar{u}_1 = 0$  in every dual optimal solution  $\bar{u}$  by Theorem 2.33. By hypothesis there is an optimal solution to the linear program, so there is an  $\bar{x}$  which satisfies  $\bar{z}_0 \geq c^\top \bar{x}$  and  $(a^i)^\top \bar{x} \geq b_i$  for  $i = 2, \dots, m$ . Therefore, if there is no feasible solution to (2.28), then (2.17) of Corollary 2.21 applies (since (2.18) implies a dual optimal value greater than the primal optimal value) and there exists a nonnegative  $(\bar{u}_0, \bar{u})$  such that  $\bar{u}_1 > 0$  and  $\bar{u}_0 \bar{z}_0 \leq \bar{u}^\top b$ ,  $\bar{u}^\top A = \bar{u}_0 c^\top$ . If  $\bar{u}_0 > 0$  then  $\bar{u}/\bar{u}_0$  is an optimal dual solution with a positive first component. If  $\bar{u}_0 = 0$  then  $A^\top \bar{u} = 0$  and  $b^\top \bar{u} \geq 0$  so adding  $\bar{u}$  to any dual optimal solution gives another dual optimal solution with a positive first component.  $\square$

**Corollary 2.36** *If there is an optimal solution to*

$$\min \{c^\top x \mid a^i x \geq b_i, i = 1, \dots, m, x_i \geq 0, i = 1, \dots, n\},$$

*then there is a unique partition of the set  $\{1, \dots, n\}$  into index sets  $I, J$  such that  $I \cap J = \emptyset$  and  $I \cup J = \{1, \dots, n\}$  and for every optimal primal-dual solution  $(\bar{x}, \bar{u}, \bar{w})$  (where  $\bar{w}_i$  is the dual multiplier on constraint  $x_i \geq 0$ )*

$$\bar{x}_i > 0 \Rightarrow i \in I \text{ and } \bar{w}_i > 0 \Rightarrow i \in J, \quad i = 1, \dots, n. \quad (2.29)$$

See Greenberg [208] for a discussion of the use of partitions in the post optimal analysis of linear programs. See also Adler and Monteiro [4] and Güler et al. [218] where the emphasis is on finding a partition with interior point algorithms. It is left as an exercise to show how to find an optimal partition using projection.

**Corollary 2.37** If  $\bar{x}, \bar{u}$  is an optimal primal-dual pair for the linear program  $\min\{c^T x \mid Ax \geq b\}$ , but is not strictly complementary, then there are either alternative primal optima or alternative dual optima.

**Example 2.38** The linear program is  $\min\{x_1 + x_2 \mid x_1 + x_2 \geq 1, x_1, x_2 \geq 0\}$ . Solve the linear program using projection.

$$\left\{ \begin{array}{rcl} z_0 - x_1 - x_2 & \geq & 0 \quad (E0) \\ x_1 + x_2 & \geq & 1 \quad (E1) \\ x_1 & \geq & 0 \quad (E2) \\ x_2 & \geq & 0 \quad (E3) \end{array} \right\} \Rightarrow \left\{ \begin{array}{rcl} z_0 & \geq & 1 \quad (E0) + (E1) \\ z_0 & \geq & 0 \quad (E0) + (E2) + (E3) \end{array} \right\}$$

An optimal solution to this linear program is  $x_1^1 = 1, x_2^1 = 0$  and an optimal dual solution is  $u_1^1 = 1, u_2^1 = 0, u_3^1 = 0$ . Then the constraint  $x_2 \geq 0$  is not strictly complementary. The constraint  $x_2 \geq 0$  appears only in the aggregate constraint  $(E0) + (E2) + (E3)$ . This constraint has positive slack so it is possible to find an optimal primal solution which satisfies  $x_2 > 0$ . One such optimal solution is  $x_1^2 = 0, x_2^2 = 1$ . Taking a convex combination of these primal solutions using multipliers of  $1/2$  gives an optimal primal solution  $\bar{x}_1 = 1/2, \bar{x}_2 = 1/2$  and optimal dual solution  $\bar{u}_1 = 1, \bar{u}_2 = 0, \bar{u}_3 = 0$ . This primal-dual solution is strictly complementary.

The complementary slackness results of this section are stated in terms of the symmetric primal-dual pair.

**Corollary 2.39** If

$$A\bar{x} \geq b, \quad \bar{x} \geq 0 \quad (2.30)$$

$$A^T\bar{u} \leq c, \quad \bar{u} \geq 0 \quad (2.31)$$

$$(b - A\bar{x})^T\bar{u} = 0 \quad (2.32)$$

$$(c - A^T\bar{u})^T\bar{x} = 0 \quad (2.33)$$

then  $\bar{x}$  is optimal to the primal problem  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$  and  $\bar{u}$  is optimal to the dual problem  $\max\{b^T u \mid A^T u \leq c, u \geq 0\}$ .

The simplex algorithm studied in Chapter 5 works at satisfying the complementary slackness conditions. At every step, simplex works with a solution which satisfies (2.30), (2.32) and (2.33) and terminates when condition (2.31) is met. As you might suspect, it is not necessary to operate in this fashion, for example, the dual simplex algorithm, presented in Chapter 6, maintains complementary slackness and dual feasibility and works towards primal feasibility.

## 2.8 SENSITIVITY ANALYSIS

The symmetric dual to the linear program given in Example 2.15 is

$$\begin{aligned} \max \quad & 2u_1 - 8u_2 - 3u_3 - 6u_4 \\ & 0.5u_2 - 0.5u_3 + u_4 \leq 2 \\ & u_1 - u_2 + u_3 - u_4 \leq -3 \\ & u_1, u_2, u_3, u_4 \geq 0 \end{aligned}$$

The optimal solution to this dual linear program is  $u_1 = 0$ ,  $u_2 = 2$ ,  $u_3 = 0$ , and  $u_4 = 1$ .

Below is the LINDO solution to the primal problem. Also included in the LINDO output is *sensitivity analysis*. Sensitivity analysis reports are designed to provide information on how much constraint right hand sides in the linear program are allowed to change before the dual variable information is no longer valid; or, how much objective function coefficients can change and still maintain optimality of the current extreme point solution. LINDO uses the revised simplex algorithm (studied later) to find the optimal primal solution. Upon termination the simplex algorithm also finds the optimal value of the dual variables. These values are given in the *dual price* column. For minimization problems the dual prices are the negative of the optimal values of the dual variables.<sup>1</sup> The optimal dual variable value for constraint 2 (row 3) is 2 and the optimal dual variable value for constraint 4 (row 5) is 1. LINDO begins numbering with the objective function so row 2 corresponds to the first constraint.

```

MIN      2 X1 - 3 X2
SUBJECT TO
    2)  X2 >= 2
    3)  0.5 X1 - X2 >= - 8
    4) - 0.5 X1 + X2 >= - 3
    5)  X1 - X2 >= - 6
END

```

---

<sup>1</sup>Historically, the dual price was defined as the *improvement* in the objective function value per unit increase in the right hand side. Since problems were often put in the form of profit maximization improvement meant increase. But for a minimization problem, with a  $\geq$  constraint, if we increase the right hand side the problem is harder to solve. If it is harder to solve, the objective function value increases since it is a min. In this case an increase is worse, hence the negative sign.

LP OPTIMUM FOUND AT STEP 2

OBJECTIVE FUNCTION VALUE

1) -22.00000

VARIABLE	VALUE	REDUCED COST
X1	4.000000	0.000000
X2	10.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	8.000000	0.000000
3)	0.000000	-2.000000
4)	11.000000	0.000000
5)	0.000000	-1.000000

RANGES IN WHICH THE BASIS IS UNCHANGED:

VARIABLE	OBJ COEFFICIENT RANGES		
	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
X1	2.000000	1.000000	0.500000
X2	-3.000000	1.000000	1.000000

ROW	RIGHTHOOKHAND SIDE RANGES		
	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	2.000000	8.000000	INFINITY
3	-8.000000	2.000000	INFINITY
4	-3.000000	11.000000	INFINITY
5	-6.000000	INFINITY	2.000000

The optimal solution to the dual (both symmetric and non-symmetric) linear program is also provided by applying projection to the primal. Recall that this process resulted in the projected system

$$z_0 \geq -22 \quad (E0) + 2(E2) + (E4) \quad (2.34)$$

$$z_0 \geq -24 \quad (E0) + 3(E2) + 0.5(E5) \quad (2.35)$$

$$z_0 \geq -44 \quad (E0) + 4(E2) + 2(E3) + (E4) \quad (2.36)$$

$$z_0 \geq -35 \quad (E0) + 4(E2) + (E3) + 0.5(E5) \quad (2.37)$$

$$z_0 \geq -30 \quad (E0) + (E1) + 4(E2) \quad (2.38)$$

$$z_0 \geq -32 \quad (E0) + 4(E2) + (E6) \quad (2.39)$$

$$0 \geq -11 \quad (E2) + (E3) \quad (2.40)$$

Equation (2.34) is the one that is binding on the optimal objective function value. This equation results from multiplying (E1) by 0, (E2) by 2 and (E3) by 0, (E4) by 1, (E5) by 0, (E6) by 0 and then adding the result to the objective function (E0). Therefore, the maximum value of  $b^T u$  possible with  $u^T A \leq (2, -3)$  is -22. Since the dual multipliers  $u_1 = 0$ ,  $u_2 = 2$ ,  $u_3 = 0$ ,  $u_4 = 1$ ,  $u_5 = 0$  and  $u_6 = 0$  give this constraint, they must be dual optimal! In LINDO, the optimal dual variable values for the nonnegativity constraints are given by the reduced cost for each variable. More on this in Chapter 3.

Notice also that if we increase the right hand side of (E2) by 1 unit the optimal objective function value will increase by 2. This is because the multiplier on constraint (E2) in (2.34) which defines the optimal objective function value is 2. Similarly, if we increase the right hand side of constraint (E4) by 1 unit the optimal objective function value will increase by 1. Therefore, the dual variables tell how much the primal objective function value increases per 1 unit increase in the right hand side. This is consistent with the complementary slackness results in the previous section. If the slack on a  $\geq$  constraint is already positive we can increase the right hand side by at least the amount of the slack without affecting the objective function value. Since this does not change the objective function value, the dual value should be zero. If a constraint has positive slack its dual variable value is always 0 in an optimal primal-dual solution. For maximization linear programs  $\max\{c^T x \mid Ax \geq b\}$  the dual variable on a binding constraint has a nice economic interpretation. If a constraint is binding then the right hand side of the constraint is a scarce resource. The dual variable value tells the maximum amount per unit you should pay to acquire more of the scarce resource.

Of course, if we change the right hand side of a constraint too much then one of the constraints other than (2.34) might become binding and define the optimal objective function value. For example, assume we increase the right hand side of (E2). The right hand side of constraint (E2) appears with the  $z_0$  variable in (2.34) - (2.39). For every unit we increase the right hand side of constraint (E2), the right hand side of (2.34) - (2.39) increases by 2, 3, 4, 4, 4, and 4 units, respectively. Then if the right hand side of (E2) is increased enough, one of the constraints (2.35) - (2.39) becomes binding. It is also necessary to check constraint (2.40) which is  $0 \geq -11$ . Although this constraint does not involve the  $z_0$  variable, this is an important constraint because it implies that if the sum of the right hand sides of (E2) and (E3) exceeds 0, then the linear program

is not feasible. If the right hand side of (E3) is held constant, the right hand side of (E2) cannot increase by more than 11 units or the problem becomes infeasible. This is discussed further in the next subsection.

Therefore, the *allowable increase* on the right hand side coefficient of (E2) is

$$\begin{aligned} \min \left\{ \frac{-22 - (-24)}{2 - 3}, \frac{-22 - (-44)}{2 - 4}, \frac{-22 - (-35)}{2 - 4}, \frac{-22 - (-30)}{2 - 4}, \frac{-22 - (-32)}{2 - 4}, 11 \right\} \\ = \min \left\{ \frac{2}{1}, \frac{22}{2}, \frac{13}{2}, \frac{8}{2}, \frac{10}{2}, 11 \right\} = 2. \end{aligned}$$

When the right hand side of constraint (E1) is increased by 2 units then both (2.34) and (2.35) are binding. In general, if a right hand side is increased (decreased) by the amount of the allowable increase (decrease) then at least two of the constraints in (2.34) - (2.40) are binding. When at least two of these constraints are binding any further increase in the constraint right hand side will result in either the primal linear program becoming infeasible or a new dual optimal solution. In either case, the old value of 2 for the dual multiplier on the constraint  $.5x_1 - x_2 \geq -8$  is no longer optimal. Given an optimal dual solution  $\bar{u}$ , we define the *allowable increase (decrease)* on the right hand side  $b_k$  of the constraint  $(a^k)^T x \geq b_k$  as the maximum increase (decrease) allowed on  $b_k$  such that  $\bar{u}$  is still an optimal dual solution. It is important to understand that allowable increase and decrease is defined with respect to a *given set of optimal dual multipliers*. Therefore, alternative dual optima imply potentially different values for allowable increases and decreases. The allowable increase and decrease is given for each constraint right hand side in the LINDO printout. Verify these values.

Increasing the right hand side  $b_k$ , of constraint  $k$ , in the linear program

$$\min \{c^T x \mid Ax \geq b\},$$

will result in infeasibility or a new optimal dual solution. If a new optimal dual solution is the result then the new dual solution will have a strictly greater value for component  $k$ . In the terminology of the textbook by Eppen, Gould, and Schmidt [141] *hurting hurts more and more*. The following lemma is left as an exercise.

**Lemma 2.40** *If  $\bar{u}$  is an optimal dual solution to  $\min \{c^T x \mid Ax \geq b\}$  and  $\theta_k$  is the allowable increase (decrease) on the right hand side  $b_k$  then increasing (decreasing)  $b_k$  by more than  $\theta_k$  will either result in an infeasible primal or a new optimal dual solution  $\hat{u}$  where  $\hat{u}_k > \bar{u}_k$  ( $\hat{u}_k < \bar{u}_k$ ).*

Closely related to alternative dual optima is the concept of primal degeneracy. A solution  $\bar{x}$  of the primal linear program  $\min \{c^\top x \mid Ax \geq b\}$  where  $A$  is an  $m \times n$  matrix is *primal degenerate* if the submatrix of  $A$  defined by the binding constraints  $((a^i)^\top \bar{x} = b_i$  where  $a^i$  is row  $i$  of  $A$ ) has rank strictly less than the number of binding constraints. Primal degeneracy of an optimal solution is almost equivalent to alternative dual optima. In fact some authors define primal degeneracy to be alternative dual optima. In Proposition 2.41 we show that alternative dual optima implies that every optimal primal solution is degenerate.

**Proposition 2.41** *If there exist alternative optimal dual solutions to the linear program  $\min \{c^\top x \mid Ax \geq b\}$  where  $A$  is an  $m \times n$  matrix, then every optimal solution to this linear program is degenerate.*

**Proof:** Prove the contrapositive and assume that  $\bar{x}$  is an optimal non-degenerate solution to the linear program  $\min \{c^\top x \mid Ax \geq b\}$ . Show that this implies a unique optimum to the dual problem. By Corollary 2.39, if  $\bar{u} \in \mathbb{R}^m$  is an optimal solution to the dual problem  $\max \{b^\top u \mid A^\top u = c, u \geq 0\}$  then  $(a^i)^\top \bar{x} > b_i$  implies  $\bar{u}_i = 0$  where  $a^i$  is the  $i$ th row of  $A$ . Then the only components of  $\bar{u}$  which can be nonzero are those for which  $(a^i)^\top \bar{x} = b_i$ . Since  $\bar{x}$  is not degenerate the rank of the submatrix of  $A$  defined by these binding rows equal to the number of rows in the submatrix. Because there is an optimal primal solution, there is an optimal dual solution and it is the unique solution to  $A^\top u = c$  after fixing to zero the dual variables corresponding to rows which are not binding.  $\square$

When an optimal primal solution is degenerate there may be an infinite number of solutions to the system  $A^\top u = c$ . However, there may still be only one solution to this system which is also dual feasible. That is, the solution satisfies  $A^\top u = c$  and is nonnegative. Therefore, primal degeneracy does not imply alternative dual optima. Example 2.42 illustrates this phenomenon.

**Example 2.42** Consider the linear program

$$\begin{array}{lll}
 \min & -x_2 & (E0) \\
 x_1 & -x_2 & \geq -3 \quad (E1) \\
 -x_1 & -x_2 & \geq -7 \quad (E2) \\
 & -x_2 & \geq -2 \quad (E3) \\
 -2x_1 & +x_2 & \geq -2 \quad (E4) \\
 x_1 & & \geq 0 \quad (E5) \\
 x_2 & & \geq 0 \quad (E6)
 \end{array}$$

Applying projection to this linear program and eliminating the  $x$  variables gives

$$\begin{aligned}
 z_0 &\geq -2 \quad (E0) & &+(E3) \\
 z_0 &\geq -5 \quad (E0) & +(1/2)(E1) & +(1/2)(E2) \\
 z_0 &\geq -8 \quad (E0) & +2(E1) & +(E4) \\
 z_0 &\geq -7 \quad (E0) & +(E2) & +(E5) \\
 0 &\geq -2 & & (E3) & +(E6) \\
 0 &\geq -8 & 2(E1) & +(E4) & +(E6) \\
 0 &\geq -7 & & (E2) & +(E5) & +(E6) \\
 0 &\geq -10 & (E1) & +(E2) & +(E4) & +2(E6) \\
 0 &\geq -4 & & (E3) & +(E4) & +2(E5) \\
 0 &\geq -4 & (E1) & +(E2) & +2(E4) & +2(E5) \\
 0 &\geq -10 & 2(E1) & & +2(E4) & +2(E5) \\
 0 &\geq -9 & & (E2) & +(E4) & +3E(5)
 \end{aligned}$$

An optimal primal solution is  $x_1 = 2.0$  and  $x_2 = 2.0$ . This solution is not primal degenerate because the binding constraints are (E3) and (E4). The corresponding submatrix has rank 2. The unique optimal dual solution is  $\bar{u}_1 = 0$ ,  $\bar{u}_2 = 0$ ,  $\bar{u}_3 = 1$ ,  $\bar{u}_4 = 0$ ,  $\bar{u}_5 = 0$ , and  $\bar{u}_6 = 0$ . This optimal primal-dual solution is not strictly complementary. Therefore, there must be alternative primal optima. See Corollary 2.43. Below is the LINDO solution for this problem.

#### OBJECTIVE FUNCTION VALUE

1) -2.000000

VARIABLE	VALUE	REDUCED COST
X2	2.000000	.000000
X1	2.000000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	3.000000	.000000
3)	3.000000	.000000
4)	-.000000	-1.000000
5)	-.000000	.000000

NO. ITERATIONS= 0

DO RANGE(SENSITIVITY) ANALYSIS?

?y

## RANGES IN WHICH THE BASIS IS UNCHANGED:

VARIABLE	OBJ COEFFICIENT RANGES		
	CURRENT	ALLOWABLE	ALLOWABLE
	COEF	INCREASE	DECREASE
X2	-1.000000	1.000000	INFINITY
X1	.000000	.000000	INFINITY

ROW	RIGHTHOOKHAND SIDE RANGES		
	CURRENT	ALLOWABLE	ALLOWABLE
	RHS	INCREASE	DECREASE
2	-3.000000	3.000000	INFINITY
3	-7.000000	3.000000	INFINITY
4	-2.000000	2.000000	2.000000
5	-2.000000	4.000000	6.000000

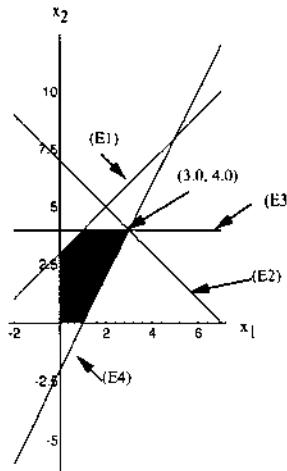
When constraint (E3) is changed to  $-x_2 \geq -4$  as illustrated in Figure 2.3 the optimal dual solution does not change and is unique; however, a new optimal primal solution is  $x_1 = 3$  and  $x_2 = 4$ . This solution is degenerate because the binding constraints are (E2), (E3) and (E4). The submatrix of A from these three constraints has rank  $2 < 3$ . In the dual problem, the variables  $u_2, u_3$  and  $u_4$  are the only dual variables which can be positive. Then the dual constraints  $A^T u = c$  reduce to

$$\begin{array}{rcl} u_2 & +2u_4 & = 0 \\ u_3 & -3u_4 & = 1 \end{array} \left. \right\} \Rightarrow \begin{array}{rcl} u_2 & = 0 & -2u_4 \\ u_3 & = 1 & +3u_4 \end{array}$$

Although there are three variables and two equations, there is only one solution to this system with the dual variables nonnegative, namely the solution  $u_2 = 0$ ,  $u_3 = 1$  and  $u_4 = 0$  which we found using projection. Therefore primal degeneracy does not imply alternative dual optima. The optimal primal-dual solution is not strictly complementary. Since there are not alternative dual optima there exist alternative primal optima. What is the other optimal primal extreme point?

Example 2.42 illustrates another interesting point. When the right hand side of (E3) is -2, the allowable decrease on the right hand side of (E3) is 3. See the equations resulting from projection. The LINDO sensitivity analysis report

**Figure 2.3** Degeneracy and Alternative Dual Optima



gives 2 for the allowable decrease on  $(E3)$ . Why not 3? Consider what happens when the right hand side of  $(E3)$  is decreased by 2 units to -4. Now when the right hand side of  $(E3)$  is decreased even further by  $\epsilon > 0$  to  $-4 - \epsilon$  a new optimal extreme point is given by the intersection of  $(E2)$  and  $(E3)$ . Refer to Figure 2.3. There is positive slack on constraint  $(E4)$ . However, the dual solution  $\bar{u}_1 = 0, \bar{u}_2 = 0, \bar{u}_3 = 1, \bar{u}_4 = 0, \bar{u}_5 = 0$ , and  $\bar{u}_6 = 0$  remains the unique optimal dual solution until  $\epsilon = 1$ . When simplex codes calculate the allowable increase and decrease they calculate how much a right hand side can change before the set of positive primal variables and constraints with positive slack change, that is before a change in the basis. See Appendix A or Chapter 5 for a definition of basis. If there are alternative primal optima, then it may be necessary to make several basis changes before the optimal dual variables change in value.

In the case of a primal-dual solution which is not strictly complementary, it would be nice to know which kind of alternative optima we have, primal or dual. This is easy if there is not primal degeneracy.

**Corollary 2.43** *If  $(\bar{x}, \bar{u})$  is an optimal primal-dual pair for the linear program  $\min \{c^T x \mid Ax \geq b\}$ , but is not strictly complementary and  $\bar{x}$  is not primal degenerate, then there exist alternative primal optima.*

Primal degeneracy implies the existence of a primal-dual pair which is not strictly complementary and therefore there are either alternative optimal dual solutions or alternative optimal primal solutions. Unfortunately in this case, it is impossible to tell from the linear programming output whether there are alternative optimal primal solutions or alternative optimal dual solutions.

**Proposition 2.44** *If  $\bar{x}$  is an optimal primal degenerate solution for the linear program  $\min\{c^T x \mid Ax \geq b\}$ , then there exists an optimal dual solution  $\bar{u}$  such that  $(\bar{x}, \bar{u})$  are not strictly complementary and there are either alternative primal optima or alternative dual optima.*

Later, in Chapter 5 all linear programs are put into the *standard form*

$$\min\{c^T x \mid Ax = b, x \geq 0\}.$$

Strict complementary slackness does not hold for the  $Ax = b$  constraints of a linear program in standard form. However, it is trivial to modify the proof of Theorem 2.35 and show that strict complementary slackness does hold for the  $x \geq 0$  constraints of a linear program in standard form.

**Proposition 2.45** *If  $(\bar{x}, \bar{u}, \bar{w})$  are primal-dual optimal for the linear program  $\min\{c^T x \mid Ax = b, x \geq 0\}$ , (where  $\bar{u}$  are the dual variables on constraints  $Ax = b$  and  $\bar{w}$  are the dual variables on constraints  $x \geq 0$ ) then there exists an optimal primal-dual solution  $(\hat{x}, \hat{u}, \hat{w})$  such that  $\hat{x}$  and  $\hat{w}$  are strictly complementary.*

Results similar to Propositions 2.41, 2.44 and Corollary 2.43 hold for a linear program in standard form. As we pointed out in Example 2.42 the conventional definition for allowable increase (decrease) is based on the simplex algorithm and the concept of basic feasible solutions. Most textbooks define the allowable increase (decrease) on the right hand side of constraint  $k$  as the largest increase (decrease) possible such that the current optimal primal basis remains optimal. If there are alternative primal optima as in Example 2.42, this may underestimate the true allowable increase and decrease as we define it. In Proposition 6.1 in Chapter 6, we show that if the optimal dual solution is not degenerate then the simplex algorithm will give the actual allowable increase and decrease for the right hand side coefficients.

In summary, if  $(\bar{x}, \bar{u})$  is an optimal primal-dual pair, but are not strictly complementary, then there are alternative primal optima, or alternative dual optima. If there are alternative dual optima, every primal solution is degenerate (but

the converse is not always true). If there are alternative primal optima, every dual solution is degenerate (again, the converse is not necessarily true). A dual solution  $\bar{u}$  is *dual degenerate* if the rank of the matrix defined by the constraints  $A^T \bar{u} = c$  plus the nonnegativity constraints for which  $\bar{u}_i = 0$  is not full row rank. See Exercise 3.10. When we use the term degenerate without a primal or dual qualification we mean lack of a strictly complementary solution.

In the next chapter we study the allowable increase and decrease on the objective function and observe that for a linear program, decreasing (increasing) the objective function coefficient of a variable by more than its allowable decrease (increase) makes the current primal optimal solution nonoptimal. Our definition of allowable increase (decrease) on the right hand side coefficient as being the maximum change allowed so that the current optimal dual solution remains optimal is symmetric with objective function coefficient range analysis. We prefer a definition of allowable increase (decrease) which is independent of the concept of basic feasible solutions and the simplex algorithm. Also, the amount we can change a right hand side coefficient before the optimal dual variables change is what we really care about, that is, the range for which the current values of the dual variables are valid. The differences with simplex sensitivity analysis are addressed further in the Chapter 6.

### 2.8.1 The Optimal Value Function

Define the function  $z_0(b) = \min \{c^T x \mid Ax \geq b\}$ . This function is the *optimal value function* of a linear program.

**Example 2.46 (Example 2.15 Continued)** Let  $b_1, \dots, b_6$  denote arbitrary right hand sides for the linear program in Example 2.15. From system (2.34) - (2.39) it follows that

$$z_0(b) = \max \{2b_2 + b_4, 3b_2 + 0.5b_5, 4b_2 + 2b_3 + b_4, 4b_2 + b_3 + 0.5b_5, b_1 + 4b_2, 4b_2 + b_6\}$$

Thus, the optimal value function for this linear program is piecewise linear. Equation (2.40) is  $0 \geq -11$  (E2) + (E3) and is not part of the optimal value function. However, any equation of the form  $0 \geq d_k$  is a feasibility condition. The linear program cannot be feasible if  $d_k > 0$ . This example linear program is feasible if and only if  $b_2 + b_3 \leq 0$ . This example illustrates two important general results that hold for linear programs.

**Proposition 2.47 (Feasibility)** There exists a piecewise linear function  $\hat{z}_0 : \mathbb{R}^m \rightarrow \mathbb{R}^1$  such that  $P = \{x \mid Ax \geq b\} \neq \emptyset$  if and only if  $\hat{z}_0(b) \leq 0$ .

**Proposition 2.48 (Optimal Value Function)** *If the projection of the system  $z_0 - c^\top x, Ax \geq b$  into the  $z_0$  variable space is not  $\mathbb{R}^1$ , and is not null, then there exists a set of vectors,  $u^1, \dots, u^q$  such that*

$$z_0(b) = \min \{c^\top x \mid Ax \geq b\} = \max \{b^\top u^k \mid k = 1, \dots, q\}$$

for all  $b$  such that  $\{x \mid Ax \geq b\} \neq \emptyset$ .

**Corollary 2.49** *The optimal value function  $z_0(b) = \min \{c^\top x \mid Ax \geq b\}$  is a piecewise linear convex function over the domain for which the linear program is feasible.*

Both of these propositions are easily proved using projection.

## 2.9 CONCLUSION

Projection, or the Fourier-Motzkin elimination method, as presented in this chapter cannot be used to solve large linear programs. It is not theoretically good, (see Exercise 2.3) nor does it work well in practice. However, from a pedagogical standpoint we feel that theorems of the alternative, duality theory, complementary slackness and sensitivity analysis are best understood by projecting out variables. Projection is the natural extension of Gaussian elimination to linear inequalities. It is a simple and unifying concept and provides the foundation for the theory of linear inequalities. Later, in Chapters 10 and 16 we study algorithms for large integer programming problems which are based upon projection methods.

## 2.10 EXERCISES

2.1 Prove Lemma 2.10.

2.2 What is the projection of the system

$$\begin{aligned} 2x_1 + 3x_2 &\leq 12 \\ x_1 + x_2 &\leq 6 \\ x_1, x_2 &\geq 0 \end{aligned}$$

into  $x_1$  space?

- 2.3 There is no polynomial function  $f(n, m)$  such that the number of constraints which result from projecting out  $n$  variables from an  $m$  constraint system is bounded by  $f(n, m)$ . Construct an example problem with  $n$  variables and  $m = 2^n$  constraints, such that projection generates

$$(1/2)^{(2^{n+1}-2)}m^{2^n}$$

inequalities in the projected system.

- 2.4 Use projection to solve the following linear program.

$$\begin{aligned} & \max 3x_1 + 9x_2 \\ \text{s.t. } & 2x_1 + 3x_2 \leq 12 \\ & x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

- 2.5 Write out the dual for the following linear program.

$$\begin{aligned} & \max 3x_1 + 9x_2 \\ \text{s.t. } & 2x_1 + 3x_2 \leq 12 \\ & x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

- 2.6 Write out the dual for the following linear program.

$$\begin{aligned} & \min 3x_1 + 9x_2 \\ \text{s.t. } & 2x_1 + 3x_2 \leq 12 \\ & x_1 + x_2 \leq 6 \\ & x_1 + 2x_2 = 4 \\ & x_2 \geq 0 \end{aligned}$$

There is not a nonnegative constraint on variable  $x_1$ .

- 2.7 You are given a linear program which is *unbounded*. Clearly describe how you could test for this by using projection.
- 2.8 If  $P = \{x | Ax \leq b\} \neq \emptyset$  is a *polytope* in  $\mathbb{R}^n$  show how projection can be used to find an *extreme point* of  $P$ . What problem might arise when  $P$  is a polyhedron?

- 2.9 Consider the *single product dynamic lot size model*. Let  $x_t$  denote production in period  $t$  for  $t = 1, \dots, T$ ,  $I_t$  is the units of inventory held at the end of period  $t$ ,  $c_t$  the marginal production cost,  $h_t$  the marginal holding cost and  $d_t$  the demand in period  $t$ . The model is

$$\begin{aligned} \min \quad & \sum_{t=1}^T (c_t x_t + h_t I_t) \\ \text{s.t.} \quad & I_{t-1} + x_t - I_t = d_t, \quad t = 1, \dots, T \\ & x_t, I_t \geq 0, \quad t = 1, \dots, T \end{aligned}$$

- a. Let  $u_t$  be the dual variable on the  $t$ th demand constraint. Write the dual problem for the model above.
  - b. Prove that the optimal dual variables are given by
- $$u_t^* = \min\{c_t, h_{t-1} + u_{t-1}^*\} \quad t = 1, \dots, T,$$
- where  $u_0^* = 0$ . Your proof should also involve constructing the optimal primal solution.
- c. Give an economic interpretation to the  $u_t^*$ .

2.10 Prove Corollary 2.21.

2.11 Given polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  where  $A$  is an  $m \times n$  matrix and  $b$  an  $m$  vector, let  $J(x_1) = \{i \mid a_{i1} > 0\}$ . If  $J(x_1)$  has cardinality of at least 1 and  $a_{i1} \geq 0$ ,  $i = 1, \dots, m$ , what is the projection into the sub-space defined by the  $x_2, \dots, x_n$  variables?

2.12 Prove that if  $P = \{x \mid Ax \geq b\}$  is a polytope then it is a polyhedron.

2.13 Use Corollary 2.19 to prove Weyl's theorem.

2.14 Prove that the affine hull of a finite set of points in  $\mathbb{R}^n$  is a polyhedron.

2.15 Consider the following system of inequalities.

$$\begin{aligned} x_1 - 2x_2 + x_3 + 6x_4 &\geq 6 \\ -2x_1 + 3x_2 + 4x_3 + x_4 &\geq 5 \\ 4x_1 - 4x_2 + x_3 + x_4 &\geq 10 \\ x_1 + 2x_2 &\geq 0 \\ x_2 &\geq 0 \\ x_3 &\geq 0 \\ x_4 &\geq 0 \end{aligned}$$

Find the projection of this system into  $(x_3, x_4)$  space. Which of the inequalities in  $(x_3, x_4)$  space correspond to extreme rays of the projection cone?

2.16 Find the allowable increase and decrease for the right hand side coefficients for constraints  $2x_1 + 3x_2 \leq 12$  and  $x_1 + x_2 \leq 6$  in Exercise 2.4.

2.17 Prove that the optimal value function

$$z_0(b) = \min\{c^\top x \mid Ax \geq b\}$$

is a piecewise-linear convex function.

2.18 Construct an example of a linear program with a non-zero objective function that has a degenerate primal optimal solution, but the optimal value of the dual variable associated with one of the binding constraints does not change for either small increases or decreases in the right hand side value of the constraint.

2.19 If the linear program  $\min\{c^\top x \mid Ax \geq b\}$  has an optimal solution show that there is a nonnegative  $\bar{u}$  such that  $A^\top \bar{u} = c$  and the positive components of  $\bar{u}$  correspond to linearly independent rows of  $A$ .

2.20 Let  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$  and  $\text{proj}_y(P)$  be the projection of  $P$  into the sub-space of  $y$ . Construct an example where  $\bar{u}$  is an extreme ray of the projection cone

$$C_x(P) = \{u \mid A^\top u = 0, u \geq 0\}$$

but  $(\bar{u}^\top B)y \geq \bar{u}^\top b$  is not needed to describe  $\text{proj}_y(P)$ .

2.21 Show that if  $\bar{u}$  is an optimal solution to the dual of the linear program  $\min\{c^\top x \mid Ax \geq b\}$  then an optimal primal solution to this linear program is contained in the solution set to the system  $A_{i,\bullet}x = b_i$  for all  $i$  such that  $\bar{u}_i = 0$ .

2.22 If the system  $Ax \geq b$  is inconsistent then a subsystem  $A'x \geq b'$  of  $Ax \geq b$  is an *irreducible inconsistent system* (IIS) if it is inconsistent but deleting any constraint from  $A'x \geq b'$  results in a feasible system. Prove or provide a counter example to the following proposition. If  $Ax \geq b$  is inconsistent and  $\bar{u}$  is an extreme ray of the cone  $C = \{u \mid A^\top u = 0, u \geq 0\}$  such that  $\bar{u}b > 0$ , then the subsystem of  $Ax \geq b$  defined by the constraints corresponding to  $\bar{u}_i > 0$  is an IIS.

2.23 Find the equations describing the polytope that is the convex hull of the points  $(0, 0, 0)$ ,  $(0, -1, 1)$ ,  $(2, 0, 2)$  and  $(-1, 1, 4)$ .

- 2.24 Prove that if  $(\bar{x}, \bar{u})$  is an optimal primal-dual pair for the linear program  $\min \{cx \mid Ax \geq b\}$ , but is not strictly complementary and  $\bar{x}$  is not degenerate, then there are alternative primal optima.
- 2.25 If there is an optimal solution to the linear program  $\min\{cx \mid Ax = b, x \geq 0\}$  is there an optimal strictly complementary solution?
- 2.26 Prove that if  $\bar{x}, \bar{u}$  is an optimal primal-dual pair for the linear program  $\min \{c^T x \mid Ax \geq b\}$ , but is not strictly complementary, then there are either alternative primal optima or alternative dual optima.
- 2.27 Consider the *knapsack problem*:

$$\min \left\{ \sum_{i=1}^n c_i x_i \mid \sum_{i=1}^n a_i x_i \geq b, x_i \geq 0 \right\}.$$

Prove that an optimal solution to the knapsack problem can be found using projection and that the number of arithmetic operations is bounded by a polynomial function of  $n$ .

- 2.28 Prove Farkas' lemma using Corollary 2.27.
- 2.29 Prove Proposition 2.47.
- 2.30 Prove Proposition 2.48.
- 2.31 Prove that if the linear program  $\min \{c^T x \mid Ax \geq b\}$  has an optimal solution for  $b = \bar{b}$  then the linear program is never unbounded for any value of  $b$ .
- 2.32 Prove that applying projection to the linear program  $\min\{c^T x \mid Ax \geq b\}$  will yield all of the extreme points of the dual polyhedron  $\{u \mid A^T u = c, u \geq 0\}$ .
- 2.33 Explain how you can apply projection to the primal problem,  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$ , and recognize that the dual of this problem is unbounded.
- 2.34 Explain how you can apply projection to the primal problem,  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$ , and find the optimal partition of the variables described in Corollary 2.36.
- 2.35 Consider the linear program  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$ . If the allowable increase on at least one of the nonnegativity constraints is positive, does this mean there are alternative primal optimal solutions?

- 2.36 *The 100% Percent Rule.* Show that if the sum of the percentage increase (decrease) (that is the increase (decrease) in the right hand side divided by the allowable increase (decrease) times 100 ) of all the right hand sides in the linear program,  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$ , does not exceed 100% then the current optimal dual solution remains optimal. See Bradley, Hax, and Magnanti [72].
- 2.37 Consider the linear program  $\min\{c^T x \mid Ax = b, Bx \geq d\}$ . Explain how to find the optimal dual variables on the constraints  $Ax = b$  through a variable elimination procedure.
- 2.38 Prove Proposition 2.44.
- 2.39 Construct an example where both the primal *and* dual linear program are infeasible.
- 2.40 Before optimizing the linear program  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$  using the simplex algorithm (studied in Chapter 5) it is necessary to find a feasible solution. A feasible solution is found by solving the linear program

$$\begin{aligned} \min \quad & x_0 \\ a^i x + x_0 & \geq b_i, \quad i = 1, \dots, m \\ x, x_0 & \geq 0 \end{aligned}$$

where  $x_0$  is an artificial variable. Prove that if the optimal solution to this LP is strictly positive, then there is an optimal dual solution such that the positive dual variables for  $Ax \geq b, x \geq 0$  index an (IIS).

- 2.41 Consider the primal linear program  $\min\{c^T x \mid Ax = b, x \geq 0\}$  and its dual  $\max\{b^T u \mid A^T u + w = c, w \geq 0\}$ . Assume that the primal problem is feasible. Prove that the set of optimal solutions to the primal is non-empty and bounded if and only if there is a strictly feasible solution  $(\bar{u}, \bar{w})$  to the dual, i.e.  $\bar{w} > 0$ .
- 2.42 Construct an example of a linear program in  $\mathbb{R}^2$  which has an optimal degenerate primal solution but does not have any redundant constraints.
- 2.43 Prove Lemma 2.40.
- 2.44 Prove Proposition 2.45.

---

# LINEAR SYSTEMS AND INVERSE PROJECTION

## 3.1 INTRODUCTION

The projection process replaces variables with constraints. In this chapter, we study the inverse, or dual of projection, and *replace constraints with variables*. Our development is based primarily on Dantzig and Eaves [109] and Williams [453]. Replacing constraints with variables is illustrated next in Section 3.2. We refer to the process of replacing constraints with variables as *inverse projection*. This is logical because in projection we replace variables with constraints. In Section 3.2 we also show how to solve linear programs with inverse projection and give a simple proof of the finite basis theorem for polyhedra. In Section 3.3 we further illustrate the duality between projection and inverse projection. Sensitivity analysis of the objective function coefficients is covered in Section 3.4. Concluding remarks are given in Section 3.5. Exercises are provided in Section 3.6.

## 3.2 DELETING CONSTRAINTS BY ADDING VARIABLES

Consider the following example which is a continuation of Example 2.15 from Chapter 2.

### Example 3.1

$$x_2 \geq 2$$

$$\begin{aligned}
 0.5x_1 - x_2 &\geq -8 \\
 -0.5x_1 + x_2 &\geq -3 \\
 x_1 - x_2 &\geq -6 \\
 x_1, x_2 &\geq 0
 \end{aligned}$$

We "standardize" this system as follows.

$$\begin{array}{ccccccc}
 & x_2 & -2x_0 & -x_3 & & = & 0 \\
 0.5x_1 & -x_2 & +8x_0 & & -x_4 & = & 0 \\
 -0.5x_1 & +x_2 & +3x_0 & & -x_5 & = & 0 \\
 x_1 & -x_2 & +6x_0 & & -x_6 & = & 0 \\
 & & x_0 & & & & = 1 \\
 x_1, & x_2, & x_0, & x_3, & x_4, & x_5 & x_6 \geq 0
 \end{array}$$

In this example the variables are nonnegative. If the variables were not non-negative then one could eliminate constraints (and variables) using Gaussian elimination. We are really doing a "dual" algorithm and nonnegative variables correspond to inequalities as opposed to equalities in the primal. After eliminating the first constraint we will explain why the inequality constraints were converted to equality constraints by subtracting off the slack variables  $x_3, x_4, x_5$  and  $x_6$ .

The objective is to find a transformation of variables so that the first constraint simplifies to  $0 = 0$  and can be deleted. This is done by creating a new variable for each combination of variables with positive and negative coefficients. Since there is one variable  $x_2$ , with a positive coefficient, and two variables  $x_0, x_3$  with negative coefficients we define two new variables,  $y_1$  and  $y_2$ . The construction in the table below shows how to generate the coefficients of the new variables.

		$y_1$	$y_2$
1	$x_2$	1	1
-2	$x_0$	0.5	
-1	$x_3$		1

This leads to the variable transformation:

$$\begin{aligned}
 x_2 &= y_1 + y_2 \\
 x_0 &= 0.5y_1 \\
 x_3 &= y_2
 \end{aligned}$$

Using this transformation gives the system

$$\begin{array}{ccccccccc} & (y_1 + y_2) & -y_1 & -y_2 & & & & = & 0 \\ 0.5x_1 & -(y_1 + y_2) & +4y_1 & & -x_4 & & & = & 0 \\ -0.5x_1 & +(y_1 + y_2) & +1.5y_1 & & & -x_5 & & = & 0 \\ x_1 & -(y_1 + y_2) & +3y_1 & & & & -x_6 & = & 0 \\ & & 0.5y_1 & & & & & = & 1 \\ x_1, & & y_1, & y_2, & x_4, & x_5, & x_6 & \geq & 0 \end{array}$$

Since the  $x$  variables in the original system are nonnegative the  $y_i$  must also be nonnegative. This system then simplifies to (we drop the  $0 = 0$  constraint)

$$\begin{array}{ccccccccc} 0.5x_1 & +3y_1 & -y_2 & -x_4 & & & & = & 0 \\ -0.5x_1 & +2.5y_1 & +y_2 & & -x_5 & & & = & 0 \\ x_1 & +2y_1 & -y_2 & & & -x_6 & & = & 0 \\ & 0.5y_1 & & & & & & = & 1 \\ x_1, & y_1, & y_2, & x_4, & x_5, & x_6 & \geq & 0 & \end{array}$$

Why was the original system converted from inequalities to equalities? If this had not been done, then the first constraint  $x_2 - 2x_0 \geq 0$  could be eliminated by the simple variable substitution  $x_2 = y_1$ , and  $x_0 = 0.5y_1$ . Then any setting of  $y_1$  gives a value of 0 for  $x_2 - 2x_0$ . This is too restrictive. This value can be positive, i.e. the constraint has slack. By adding the variable  $x_3$  and then doing the variable substitution we allow for  $x_3 > 0$ , i.e. the constraint has slack.

Now eliminate the constraint  $0.5x_1 + 3y_1 - y_2 - x_4 = 0$  through a variable transformation.

		$v_1$	$v_2$	$v_3$	$v_4$
0.5	$x_1$	2	2		
3	$y_1$			1/3	1/3
-1	$y_2$	1		1	
-1	$x_4$		1		1

The appropriate variable transformation is then

$$\begin{aligned} x_1 &= 2v_1 + 2v_2 \\ y_1 &= (1/3)v_3 + (1/3)v_4 \\ y_2 &= v_1 + v_3 \\ x_4 &= v_2 + v_4 \end{aligned}$$

Using this variable transformation gives the system

$$-v_2 + (11/6)v_3 + (5/6)v_4 - x_5 = 0 \quad (3.1)$$

$$v_1 + 2v_2 - (1/3)v_3 + (2/3)v_4 - x_6 = 0 \quad (3.2)$$

$$(1/6)v_3 + (1/6)v_4 = 1 \quad (3.3)$$

$$v_1, v_2, v_3, v_4, x_5, x_6 \geq 0 \quad (3.4)$$

Of course, the original variables can also be expressed in terms of the  $v$  variables as

$$\begin{aligned} x_1 &= 2v_1 + 2v_2 \\ x_2 &= v_1 + (4/3)v_3 + (1/3)v_4 \\ x_3 &= v_1 + v_3 \\ x_4 &= v_2 + v_4 \\ x_5 &= x_5 \\ x_6 &= x_6 \end{aligned}$$

Next eliminate the constraint  $-v_2 + (11/6)v_3 + (5/6)v_4 - x_5 = 0$  through a variable transformation.

		$w_1$	$w_2$	$w_3$	$w_4$
$-1$	$v_2$	1	1		
$(11/6)$	$v_3$	$(6/11)$			$(6/11)$
$(5/6)$	$v_4$		$(6/5)$	$(6/5)$	
$-1$	$x_5$			1	1

The corresponding variable transformation is

$$\begin{aligned} v_2 &= w_1 + w_2 \\ v_3 &= (6/11)w_1 + (6/11)w_4 \\ v_4 &= (6/5)w_2 + (6/5)w_3 \\ x_5 &= w_3 + w_4 \end{aligned}$$

Substituting these variables into the system (3.1)-(3.4) gives:

$$v_1 + (20/11)w_1 + (14/5)w_2 + (4/5)w_3 - (2/11)w_4 - x_6 = 0 \quad (3.5)$$

$$(1/11)w_1 + (1/5)w_2 + (1/5)w_3 + (1/11)w_4 = 1 \quad (3.6)$$

$$v_1, w_1, w_2, w_3, w_4, x_6 \geq 0 \quad (3.7)$$

Transforming back to the original variables gives

$$\begin{aligned}
 x_1 &= 2v_1 + 2w_1 + 2w_2 \\
 x_2 &= v_1 + (8/11)w_1 + (2/5)w_2 + (2/5)w_3 + (8/11)w_4 \\
 x_0 &= (1/11)w_1 + (1/5)w_2 + (1/5)w_3 + (1/11)w_4 \\
 x_3 &= v_1 + (6/11)w_1 + (6/11)w_4 \\
 x_4 &= w_1 + (11/5)w_2 + (6/5)w_3 \\
 x_5 &= w_3 + w_4 \\
 x_6 &= x_6
 \end{aligned}$$

Notice that variable  $w_1$  has five positive coefficients in the transformation matrix. Since only three constraints have been eliminated this means that  $w_1$  is not necessary and can be dropped. Why? Refer back to Corollary 2.25 in Section 2.5 in Chapter 2 and note the analogy with dropping constraints when projecting out variables.

Finally, eliminate the constraint  $v_1 + (14/5)w_2 + (4/5)w_3 - (2/11)w_4 - x_6 = 0$  through a variable transformation.

		$z_1$	$z_2$	$z_3$	$z_4$	$z_5$	$z_6$
1	$v_1$	1	1				
$14/5$	$w_2$			$5/14$	$5/14$		
$4/5$	$w_3$					$5/4$	$5/4$
$-2/11$	$w_4$	$11/2$		$11/2$		$11/2$	
-1	$x_6$		1		1		1

The corresponding variable transformation is

$$\begin{aligned}
 v_1 &= z_1 + z_2 \\
 w_2 &= (5/14)z_3 + (5/14)z_4 \\
 w_3 &= (5/4)z_5 + (5/4)z_6 \\
 w_4 &= (11/2)z_1 + (11/2)z_3 + (11/2)z_5 \\
 x_6 &= z_2 + z_4 + z_6
 \end{aligned}$$

Substituting these variables into the system (3.5)- (3.7) gives

$$(1/2)z_1 + (4/7)z_3 + (1/14)z_4 + (3/4)z_5 + (1/4)z_6 = 1 \quad (3.8)$$

$$z_1, z_2, z_3, z_4, z_5, z_6 \geq 0 \quad (3.9)$$

Transforming back to the original variables gives:

$$\begin{aligned}
 x_1 &= 2z_1 + 2z_2 + (5/7)z_3 + (5/7)z_4 \\
 x_2 &= 5z_1 + z_2 + (29/7)z_3 + (1/7)z_4 + (9/2)z_5 + (1/2)z_6 \\
 x_0 &= (1/2)z_1 + (4/7)z_3 + (1/14)z_4 + (3/4)z_5 + (1/4)z_6 \\
 x_3 &= 4z_1 + z_2 + 3z_3 + 3z_5 \\
 x_4 &= (11/14)z_3 + (11/14)z_4 + (3/2)z_5 + (3/2)z_6 \\
 x_5 &= (11/2)z_1 + (11/2)z_3 + (27/4)z_5 + (5/4)z_6 \\
 x_6 &= z_2 + z_4 + z_6
 \end{aligned} \tag{3.10}$$

In Figure 3.1 we use this transformation to plot the points in  $(x_1, x_2)$  space that correspond to the extreme points of (3.8)- (3.9). In this example a  $z_1 = 2$ , all other  $z_i = 0$  corresponds to the extreme point solution  $x_1 = 4$  and  $x_2 = 10$ . The solution  $z_4 = 14$  and all other  $z_i = 0$  corresponds to the extreme point  $x_1 = 10$  and  $x_2 = 2$ . The solution  $z_5 = 4/3$  and all other  $z_i = 0$  corresponds to the extreme point  $x_1 = 0, x_2 = 6$ . The solution  $z_6 = 4$  and all other  $z_i = 0$  corresponds to the extreme point  $x_1 = 0, x_2 = 2$ . The solution  $z_3 = 7/4$  and all other  $z_i = 0$  corresponds to the feasible point  $(5/4, 29/4)$ , which is not an extreme point. Thus, any nonnegative feasible solution to (3.8) - (3.9) corresponds to a feasible point in the original polyhedron and we can generate any extreme point of the polyhedron! The solution  $z_2 = 1$  corresponds to the extreme ray  $x_1 = 2$  and  $x_2 = 1$  of the recession cone of the polyhedron. Thus, any of the generators of the original polyhedron can be generated by appropriate choice of the  $z_i$ . This implies that if the  $z_i$  were projected out of the system (3.8)-(3.10) the resulting subspace in the  $x_0, x_1, \dots, x_6$  variables would correspond to the original polyhedron. This is shown formally in Proposition 3.2.

We now give a formal description of what we have done in the example. Apply inverse projection to the system

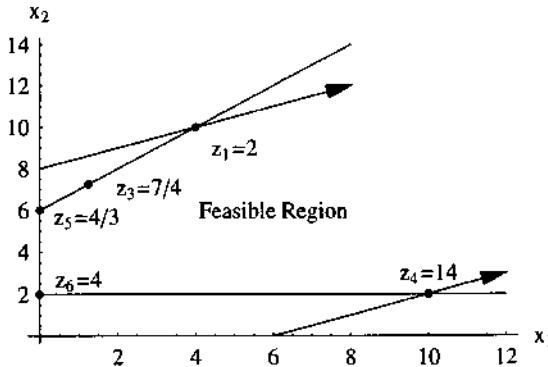
$$Ax - bx_0 = 0, \quad x_0 = 1, \quad x \geq 0. \tag{3.11}$$

Begin with row one. Assume the columns are ordered so that  $a_{1i} > 0$  for  $i = 1, \dots, n_1$ ,  $a_{1,n_1+j} < 0$ , for  $j = 1, \dots, n_2$  and  $a_{1l} = 0$ , for  $l = n_1 + n_2 + 1, \dots, n$ . Also, assume without loss,  $b_1 < 0$ . The first constraint  $\sum_{j=1}^n a_{1j}x_j - b_1x_0 = 0$  in (3.11) is eliminated using the variable transformation

$$x_i = \sum_{j=1}^{n_2} y_{ij}/a_{1i}, \quad i = 1, \dots, n_1, \quad x_0 = \sum_{j=1}^{n_2} y_{0j}/b_1. \tag{3.12}$$

$$x_{n_1+j} = -\sum_{i=0}^{n_1} y_{ij}/a_{1,n_1+j}, \quad j = 1, \dots, n_2. \tag{3.13}$$

Figure 3.1 Points of Transformation



Now substitute out the  $x, x_0$  variables in the system  $Ax - bx_0 = 0, x_0 = 1$  using (3.12)-(3.13). The new system is

$$\begin{aligned} \sum_{i=1}^{n_1} \left( a_{ki} \sum_{j=1}^{n_2} y_{ij} / a_{1i} \right) - \sum_{j=1}^{n_2} \left( a_{k, n_1+j} \sum_{i=0}^{n_1} y_{ij} / a_{1, n_1+j} \right) \\ + \sum_{l=n_1+n_2+1}^n a_{kl} x_l - b_k \sum_{j=1}^{n_2} y_{0j} / b_1 = 0, \quad k = 1, \dots, m \quad (3.14) \end{aligned}$$

$$\sum_{j=1}^{n_2} y_{0j} / b_1 = 1. \quad (3.15)$$

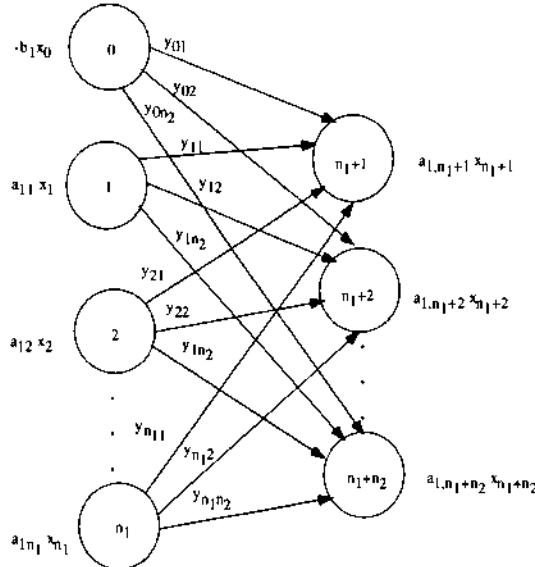
Nonnegativity of  $x$  is guaranteed by requiring  $y$  and  $x_i, i = n_1 + n_2 + 1, \dots, n$  to be nonnegative.

$$y_{ij} \geq 0, \quad i = 0, \dots, n_1, \quad j = 1, \dots, n_2, \quad x_i \geq 0, \quad i = n_1 + n_2 + 1, \dots, n. \quad (3.16)$$

**Proposition 3.2** If  $P = \{(x, x_0) \in \Re^{n+1} \mid Ax - bx_0 = 0, x_0 = 1, x \geq 0\}$  then  $(\bar{x}, \bar{x}_0) \in P$  if and only if there is a  $\bar{y}$  such that  $(\bar{x}, \bar{x}_0, \bar{y})$  is feasible to (3.12)-(3.16).

**Proof:** First show that the projection of (3.12)-(3.16) onto the  $x$ -variable subspace is  $P$ . Assume  $(\bar{x}, \bar{x}_0, \bar{y})$  is a feasible solution to (3.12)-(3.16). Show this implies  $(\bar{x}, \bar{x}_0) \in P$ . Aggregate the constraints in (3.12) using the multipliers

**Figure 3.2** Transportation Network For inverse Projection



$a_{ki}$ , for  $i = 1, \dots, n_1$ ,  $-b_k$  for  $i = 0$ , and the constraints in (3.13) using  $a_{k,n_1+j}$ , for  $j = 1, \dots, n_2$ . Then row  $k$  of (3.14) implies row  $k$  of  $Ax - bx_0 = 0$  is satisfied. Equation (3.13) and (3.15) imply  $x_0 = 1$ . The nonnegativity of  $(\bar{x}, \bar{x}_0)$  follows from (3.16).

Next assume  $(\bar{x}, \bar{x}_0) \in P$ . The crucial idea is to show that there is a feasible solution to (3.12)-(3.13) for  $(x, x_0)$  fixed at  $(\bar{x}, \bar{x}_0)$ . At this point, we refer the reader to Subsection 1.3.5 in Chapter 1 where the transportation linear program, (*TLP*) is introduced. Set up a transportation linear program with supply nodes  $i = 1, \dots, n_1$  with a supply of  $a_{1i}\bar{x}_i$  and a single supply node  $i = 0$  with supply  $-b_1\bar{x}_0$ . There are demand nodes  $j = 1, \dots, n_2$  with demand  $-a_{1,n_1+j}\bar{x}_{n_1+j}$  for  $j = 1, \dots, n_2$ . This transportation network is illustrated in Figure 3.2. Since total supply equal total demand, there is a feasible solution  $\bar{y}$  to this linear program. Thus,  $(\bar{x}, \bar{x}_0, \bar{y})$  is feasible to (3.12)-(3.16).  $\square$

In words, Proposition 3.2 states the  $P$  is the result of projecting out the  $y$  variables from (3.12)-(3.16).

### 3.2.1 The Finite Basis Theorem

Every polyhedron  $P$ , can be expressed as

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}),$$

where the  $x^i$ ,  $i = 1, \dots, r$  are points in  $P$ . This is an example of a finite basis theorem and these theorems play a key role in linear and integer linear optimization. This result for polyhedra is due to Minkowski [335]. An inductive proof of the finite basis theorem was given in Appendix A. Here we provide a much easier proof using inverse projection.

**Theorem 3.3 (Finite Basis Theorem for Polyhedra (Minkowski))** *If  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \neq \emptyset$ , then  $P$  is finitely generated. In particular, there exist  $x^1, \dots, x^q, x^{q+1}, \dots, x^r$  in  $P$  such that*

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}).$$

*Furthermore, the extreme points of  $P$  are contained in the set  $\{x^1, \dots, x^q\}$  and the extreme rays of the recession cone  $\{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$  are contained in the set  $\{x^{q+1}, \dots, x^r\}$ .*

**Proof:** Repeated application of inverse projection to the system

$$Ax - bx_0 = 0, \quad x_0 = 1, \quad x \geq 0 \quad (3.17)$$

results in an equivalent system in a new set of variables  $z_1, \dots, z_r$ ,

$$d_1 z_1 + \dots + d_q z_q = 1 \quad (3.18)$$

$$z_1, \dots, z_q, z_{q+1}, \dots, z_r \geq 0. \quad (3.19)$$

The inverse projection method also produces an  $(n+1) \times (q+r)$  transformation matrix  $T$  relating  $(x, x_0)$  and  $z$  by

$$\begin{bmatrix} x \\ x_0 \end{bmatrix} = Tz. \quad (3.20)$$

By Proposition 3.2,  $(\bar{x}, \bar{x}_0)$  is a solution of (3.17) if and only if there is an  $\bar{z}$  such that  $(\bar{x}, \bar{x}_0, \bar{z})$  is a solution of (3.18)-(3.20).

The solutions of (3.18)-(3.19) are easily characterized. If  $e^i$  is an  $r$ -component unit vector with component  $i$  equal to 1, then the extreme points are  $z^i =$

$(1/d_i)e^i$  for  $i = 1, \dots, q$  and the extreme rays are  $z^i = e^i$ , for  $i = q+1, \dots, r$ . Let  $\hat{T}$  be the matrix obtained by deleting the last row of  $T$  (the row corresponding to  $x_0$ ). If  $x^i = \hat{T}z^i$ ,  $i = 1, \dots, r$ , then

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}).$$

An extreme point of  $P$  cannot be written as a convex combination of other distinct points in  $P$ . Therefore, the extreme points of  $P$  are contained in the set  $\{x^1, \dots, x^q\}$ . Similarly, the extreme rays of the recession cone  $\{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$  are contained in the set  $\{x^{q+1}, \dots, x^r\}$ .  $\square$

A direct result of the finite basis theorem is the *fundamental theorem of linear programming*, which states that if there is an optimal solution to a linear program there is an optimal extreme point solution to the linear program.

**Theorem 3.4 (Fundamental Theorem of Linear Programming)** *If the linear program  $\min \{c^\top x \mid Ax = b, x \geq 0\}$  has an optimal solution, then there is an optimal extreme point solution to this linear program. If there is not an optimal solution, then the linear program is either unbounded or infeasible.*

**Proof:** By Minkowski's theorem, if  $P$  is not empty there exist  $x^1, \dots, x^q, x^{q+1}, \dots, x^r$  in  $\mathbb{R}^n$  such that

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}).$$

Without loss, assume the set  $\{x^1, \dots, x^q\}$  is minimal with respect to taking subsets. Then each  $x^1, \dots, x^q$  is an extreme point of  $P$ . If  $\bar{x} \in P$  then,

$$\bar{x} = \sum_{i=1}^r \bar{z}_i x^i, \quad \sum_{i=1}^q \bar{z}_i = 1, \quad \bar{z}_i \geq 0, \quad i = 1, \dots, r. \quad (3.21)$$

If  $\bar{x}$  is an optimal solution to the linear program, then  $c^\top x^i \geq 0$  for  $i = q+1, \dots, r$ , otherwise the linear program would be unbounded since  $\bar{x} + \alpha x^i$  is feasible for  $i = q+1, \dots, r$ . Also, (3.21) and the optimality of  $\bar{x}$  implies  $\bar{z}_i = 0$  if  $c^\top x^i > 0$  for  $i = q+1, \dots, r$  and  $c^\top x^i = c^\top \bar{x}$  if  $\bar{z}_i > 0$ ,  $i = 1, \dots, q$ . Then any  $x^i$  for which  $\bar{z}_i > 0$ ,  $i = 1, \dots, q$  is an optimal solution.  $\square$

### 3.2.2 Solving Linear Programs with Inverse Projection

We use inverse projection to solve the linear program  $\min\{c^\top x \mid Ax \geq b\}$ .

**Example 3.5** Continuing with Example 3.1 assume  $c = (2, -3, 0, 0, 0, 0, 0)^T$ . Then in the  $z$  variables this corresponds to the linear program

$$\min -11z_1 + z_2 - 11z_3 + z_4 - (27/2)z_5 - (3/2)z_6 \quad (3.22)$$

$$(1/2)z_1 + (4/7)z_3 + (1/14)z_4 + (3/4)z_5 + (1/4)z_6 = 1 \quad (3.23)$$

$$z_1, z_2, z_3, z_4, z_5, z_6 \geq 0 \quad (3.24)$$

In this linear program the objective function is  $c^T T z$  where  $T$  is defined in (3.10). Constraints (3.23)-(3.24) duplicate (3.8)-(3.9). Since  $z_2$  has positive objective function coefficient but does not appear in constraint (3.23) it is equal to zero in the optimal solution. Calculating the ratio of the objective function coefficient to the corresponding coefficient in the constraint (the bang for buck ratio) for  $z_1$  and  $z_3, \dots, z_6$  gives

$$\frac{-11}{(1/2)} = -22, \quad \frac{-11}{(4/7)} = -19.25, \quad \frac{1}{(1/14)} = 14, \quad \frac{-(27/2)}{(3/4)} = -18, \quad \frac{-(3/2)}{(1/4)} = -6.$$

Therefore the optimal solution is  $z_1 = 1$  which corresponds to an optimal  $x_1 = 4$ ,  $x_2 = 10$ . The optimal objective function value is  $-22$ .

Notice that inverse projection results in transforming the LP into a single knapsack constraint linear problem which is trivial to solve. Of course the number of variables in this linear program may be huge!

### 3.3 DUAL RELATIONSHIPS

There is a strong duality between projection and inverse projection. Given a primal problem, applying projection to its dual is equivalent to doing inverse projection on the primal. Similarly applying inverse projection to a primal problem is equivalent to doing projection on the dual problem.

**Example 3.6** Again, consider Example 3.1 with an objective function of  $\min 2x_1 - 3x_2$ . The dual of this problem is

$$\begin{aligned} \max \quad & 2u_1 - 8u_2 - 3u_3 - 6u_4 \\ & 0.5u_2 - 0.5u_3 + u_4 \leq 2 \\ & u_1 - u_2 + u_3 - u_4 \leq -3 \\ & u_1, u_2, u_3, u_4 \geq 0. \end{aligned}$$

Solve this linear program using projection. Make the objective function a constraint and the relevant system is

$$\begin{array}{rcl} -2u_1 + 8u_2 + 3u_3 + 6u_4 + z_0 & \leq & 0 \quad (E0) \\ 0.5u_2 - 0.5u_3 + u_4 & \leq & 2 \quad (E1) \\ u_1 - u_2 + u_3 - u_4 & \leq & -3 \quad (E2) \\ -u_1 & \leq & 0 \quad (E3) \\ -u_2 & \leq & 0 \quad (E4) \\ -u_3 & \leq & 0 \quad (E5) \\ -u_4 & \leq & 0 \quad (E6) \end{array}$$

The linear program resulting from eliminating  $u_1$  is

$$\begin{array}{rcl} 6u_2 + 5u_3 + 4u_4 + z_0 & \leq & -6 \quad (E0) + 2(E2) \\ -u_2 + u_3 - u_4 & \leq & -3 \quad (E2) + (E3) \\ 0.5u_2 - 0.5u_3 + u_4 & \leq & 2 \quad (E1) \\ -u_2 & \leq & 0 \quad (E4) \\ -u_3 & \leq & 0 \quad (E5) \\ -u_4 & \leq & 0 \quad (E6) \end{array}$$

Let  $y_1$  and  $y_2$  be dual variables associated with the first two rows,  $x_1$  associated with equation (E1) and  $x_4, x_5, x_6$  associated with (E4), (E5), and (E6), respectively. The objective function is  $\max z_0$ . Then the dual problem is:

$$\begin{array}{rcl} \min & +2x_1 - 6y_1 - 3y_2 & \\ & 0.5x_1 + 6y_1 - y_2 - x_4 & = 0 \\ & -0.5x_1 + 5y_1 + y_2 & - x_5 = 0 \\ & x_1 + 4y_1 - y_2 & - x_6 = 0 \\ & y_1 & = 1 \\ x_1, y_1, y_2, x_4, x_5, x_6 & \geq 0 & \end{array}$$

This is exactly the linear program that you get after eliminating the first constraint in the primal using inverse projection (after scaling the  $y^1$  column by 2 so that there is a coefficient of 1.0 in the last row)! Have you ever seen anything more beautiful?

Recall that with projection, if a column had all nonnegative (nonpositive) coefficients, then every row with a positive (negative) element in that column can be deleted when projecting out the variable. When doing inverse projection there may be rows with all nonnegative (nonpositive) coefficients. Since the right hand side is 0 and all variables are restricted to be nonnegative all the variables with positive (negative) coefficients in that row can be deleted since they must take on a zero value. This illustrates further the duality between columns and rows in projection and inverse projection.

### 3.4 SENSITIVITY ANALYSIS

Projection is used to perform right hand side range analysis. Inverse projection is used to perform objective function coefficient range analysis. This makes sense since the right hand side of the dual is defined from the objective function coefficients of the primal and as just shown projection on the dual is equivalent to inverse projection on the primal.

**Example 3.7** Consider Example 3.1. Let  $c_1$  be the objective function coefficient on variable  $x_1$  and  $c_2$  the objective function coefficient on variable  $x_2$ . If  $z_0$  is the optimal objective function value then

$$z_0 = \min \{c^T z^i \mid z^i \text{ an extreme point of (3.23)-(3.24)}\}.$$

That is,

$$z_0 = \min \{c^T z^i \mid 4c_1 + 10c_2, (5/4)c_1 + (29/4)c_2, 10c_1 + 2c_2, 0c_1 + 6c_2, 0c_1 + 2c_2\}.$$

Or,

$$\begin{aligned} z_0 &\leq 4c_1 + 10c_2 \\ z_0 &\leq (5/4)c_1 + (29/4)c_2 \\ z_0 &\leq 10c_1 + 2c_2 \\ z_0 &\leq 0c_1 + 6c_2 \\ z_0 &\leq 0c_1 + 2c_2 \end{aligned}$$

and  $z_0$  is maximized. The tight constraint in our example for  $c_1 = 2$  and  $c_2 = -3$  is  $z_0 \leq 4c_1 + 10c_2 = 4 \times 2 + 10 \times -3 = -22$ .

Using logic similar to that for right hand side sensitivity analysis, we can calculate the allowable increase on the objective function coefficient of variable  $x_1$  given  $c_1 = 2$  and  $c_2 = -3$ . We define the *allowable increase (decrease)* on the objective function coefficient of variable  $x_1$  to be the maximum increase (decrease) in  $c_1$  such the current optimal primal solution remains an optimal solution. Just as in the case with the allowable increase and decrease on the right hand side coefficients, the allowable increase and decrease on the objective function coefficients are defined with respect to a given primal solution. When an objective function coefficient is changed by its allowable increase or decrease, assuming these values are less than infinity, at least two distinct  $z_i$  are optimal, i.e. there are alternative primal optimal solutions. Note the complete primal-dual symmetry here between the allowable increase (decrease) on the

objective function coefficients and right hand side vectors. If we take the dual of the primal, then the objective function coefficients become right hand sides. Therefore, taking the dual of a problem and calculating the allowable increase and decrease on the right hand side is equivalent to calculating the allowable increase and decrease on the objective function coefficients of the primal! Similarly, alternative dual optima implies primal degeneracy and alternative primal optima implies dual degeneracy (refer back to Proposition 2.41 in Chapter 2 and see Exercise 3.10 ).

**Example 3.8** If  $c_1 = 2, c_2 = -3$  and  $c_1$  is increased the constraint  $z_0 \leq 10c_1 + 2c_2$  cannot become tight (refer back to Example 3.7) because it already has slack and a coefficient of 10 on  $c_1$  compared to 4 on the currently tight constraint. However, as  $c_1$  is increased  $z_0 \leq 0c_1 + 6c_2$  will become tight. With  $c_2$  fixed at  $-3$  we can increase  $c_1$  until  $4c_1 + 10c_2$  is equal to  $-18$ . This will happen for a  $c_1 = 3.0$ . Then the allowable increase on coefficient  $c_1$  is  $1.0$ .

Next, assume that the original objective function coefficients are  $c_1 = 5$  and  $c_2 = -3$ . In this case the linear program in the  $z$  variables is

$$\begin{aligned} \min \quad & -5z_1 + 7z_2 - (62/7)z_3 + (22/7)z_4 - (27/2)z_5 - (3/2)z_6 \\ & (1/2)z_1 + (4/7)z_3 + (1/14)z_4 + (3/4)z_5 + (1/4)z_6 = 1 \\ & z_1, z_2, z_3, z_4, z_5, z_6 \geq 0 \end{aligned}$$

and the optimal solution is  $z_5 = (4/3)$  which corresponds to an optimal solution  $x_1 = 0$  and  $x_2 = 6$  and optimal objective function value of  $-18$ . The tight constraint on the objective function is  $z_0 \leq 0c_1 + 6c_2$ . Performing the analysis on the objective function coefficients as before we see that if  $c_1 = 3$ , then

$$z_0 \leq 4c_1 + 10c_2 = (4)(3) + (10)(-3) = -18.$$

Since the current value of  $c_1$  is  $5$ , this corresponds to an allowable decrease on  $c_1$  of  $2 = 5 - 3$ . Below is the LINDO run for  $c_1 = 5$  and  $c_2 = -3$ .

```
MIN      5 X1 - 3 X2
SUBJECT TO
2]  X2 >= 2
3] .5 X1 - X2 >= - 8
4]- .5 X1 + X2 >= - 3
5]  X1 - X2 >= - 6
END
```

LP OPTIMUM FOUND AT STEP 2  
 OBJECTIVE VALUE = -18.000000

VARIABLE	VALUE	REDUCED COST
X1	.0000000	2.000000
X2	6.000000	.0000000
ROW	SLACK OR SURPLUS	DUAL PRICE
1	-18.00000	1.000000
2	4.000000	-.0000000
3	2.000000	-.0000000
4	9.000000	-.0000000
5	.0000000	-3.000000

RANGES IN WHICH THE BASIS IS UNCHANGED:

VARIABLE	OBJ COEFFICIENT RANGES		
	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
X1	5.000000	INFINITY	2.000000
X2	-3.000000	3.000000	2.000000
ROW	RIGHHAND SIDE RANGES		
	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	2.000000	4.000000	INFINITY
3	-8.000000	2.000000	INFINITY
4	-3.000000	9.000000	INFINITY
5	-6.000000	4.000000	2.000000

Linear programming codes also calculate the reduced cost for each primal variable. If the linear program is

$$\min \{ c^T x \mid Ax \geq b, x \geq 0 \}$$

and has an optimal primal-dual solution  $(\bar{x}, \bar{u}, \bar{w})$  where  $\bar{u}, \bar{w}$  are the dual variables on constraints  $Ax \geq b$  and  $x \geq 0$ , respectively, then the *reduced cost* for

the primal variable  $x_j$  in the optimal primal-dual solution  $(\bar{x}, \bar{u}, \bar{w})$  is  $c_j - \bar{u}^T A_{\bullet j}$ . If  $\bar{x}_j = 0$ , decreasing coefficient  $c_j$  by the amount of the reduced cost should produce an optimal primal solution  $\hat{x}$  where  $\hat{x}_j > 0$ . However, if the optimal dual  $\bar{u}$  is not unique, the reduced cost is not well defined. The following lemma and proposition are left as exercises.

**Lemma 3.9** *If  $(\bar{x}, \bar{u}, \bar{w})$  is an optimal primal-dual solution to the linear program  $\min \{c^T x \mid Ax \geq b, x \geq 0\}$  then the reduced cost on variable  $x_j$  is  $\bar{w}_j$ ,  $j = 1, \dots, n$ .*

**Proposition 3.10** *If  $(\bar{x}, \bar{u}, \bar{w})$  is an optimal primal-dual solution to the linear program  $\min \{c^T x \mid Ax \geq b, x \geq 0\}$  and  $\bar{x}$  is not primal degenerate, then the allowable decrease on the objective function coefficient of the primal variable  $x_j$  is equal to its reduced cost if  $\bar{x}_j = 0$ .*

These results also hold for linear programs in standard form, i.e.

$$\min \{c^T x \mid Ax = b, x \geq 0\}.$$

**Example 3.11** *Refer to the LINDO output in Example 3.8. The optimal primal solution is not degenerate. Why? The reduced cost for the objective function coefficient of variable  $x_1$  is 2.0 which is also the allowable decrease for that variable.*

Primal degeneracy makes it difficult to detect alternative primal optima. If an optimal primal solution is not degenerate the way to recognize alternative primal optima is by the existence of a primal variable  $\bar{x}_j = 0$ , and  $c_j - \bar{u} A_{\bullet j} = 0$  or an inequality constraint with zero slack and a zero dual variable, i.e an optimal primal-dual solution that is not strictly complementary. See Corollary 2.43. However, if there is primal degeneracy a non strictly complementary solution may or may not indicate alternative primal optima. The reduced cost on variable  $x_j$  is  $c_j - \bar{u} A_{\bullet j}$  where  $\bar{u}$  is a vector of optimal dual variables. For primal degenerate problems this value may be misleading. That is, decreasing a coefficient by its reduced cost does not produce an alternative primal optimal solution with variable  $x_j$  positive. Although primal degeneracy does not imply dual alternative optima, this is often the case. When there are dual alternative optima, the reduced cost is ambiguous and may not give the allowable decrease. Furthermore, as with right hand side sensitivity analysis, the allowable increase

and decrease calculated from the simplex algorithm may actually underestimate the true allowable increase and decrease. In Proposition 6.5 in Chapter 6 we show that if the primal solution is not degenerate then the simplex algorithm will give the actual allowable increase and decrease for the objective function coefficients.

**Example 3.12** The dual of the linear program in Example 2.42 is

$$\begin{array}{lllllll} \min & -3u_1 & -7u_2 & -2u_3 & -2u_4 & & \\ & u_1 & -u_2 & & -2u_4 & +w_1 & = & 0 \\ & -u_1 & -u_2 & -u_3 & +u_4 & & +w_2 & = & -1 \\ & u_1, & u_2, & u_3, & u_4, & w_1, & w_2 & \geq & 0 \end{array}$$

#### OBJECTIVE FUNCTION VALUE

1) -2.000000

VARIABLE	VALUE	REDUCED COST
U1	.000000	3.000000
U2	.000000	3.000000
U3	1.000000	.000000
U4	.000000	.000000
W1	.000000	2.000000
W2	.000000	2.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	2.000000
3)	.000000	2.000000

NO. ITERATIONS= 0

DO RANGE(SENSITIVITY) ANALYSIS?

?y

RANGES IN WHICH THE BASIS IS UNCHANGED:

VARIABLE	OBJ COEFFICIENT RANGES		
	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
U1	-3.000000	3.000000	INFINITY

U2	-7.000000	3.000000	INFINITY
U3	-2.000000	2.000000	2.000000
U4	-2.000000	4.000000	6.000000
W1	-.000000	2.000000	INFINITY
W2	-.000000	2.000000	INFINITY

ROW	RIGHHAND SIDE RANGES		
	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	.000000	.000000	INFINITY
3	-1.000000	1.000000	INFINITY

The optimal solution value of variable  $u_4$  is 0 and its reduced cost is 0. However, there are not alternative primal optima. Below is a solution printout for the same linear program but with a different dual basis used in the sensitivity report. In this discussion the word primal is referring to the linear program in this example, even though it is the dual of Example 2.42.

1) -2.000000

VARIABLE	VALUE	REDUCED COST
U1	.000000	1.000000
U2	.000000	5.000000
U3	1.000000	.000000
U4	.000000	4.000000
W1	.000000	.000000
W2	.000000	2.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	.000000
3)	.000000	2.000000

NO. ITERATIONS= 1

DO RANGE(SENSITIVITY) ANALYSIS?

?y

RANGES IN WHICH THE BASIS IS UNCHANGED:

OBJ COEFFICIENT RANGES

VARIABLE	CURRENT COEF	ALLOWABLE INCREASE	ALLOWABLE DECREASE
U1	-3.000000	1.000000	INFINITY
U2	-7.000000	5.000000	INFINITY
U3	-2.000000	2.000000	1.000000
U4	-2.000000	4.000000	INFINITY
W1	-.000000	2.000000	1.000000
W2	-.000000	2.000000	INFINITY

RIGHTHOOK SIDE RANGES			
ROW	CURRENT RHS	ALLOWABLE INCREASE	ALLOWABLE DECREASE
2	.000000	INFINITY	.000000
3	-1.000000	1.000000	INFINITY

The allowable decrease on the objective coefficient of  $u_3$  is 1. However, in the LINDO printout of the primal analysis the allowable decrease on the right hand side of constraint (E3) is 2 and there is a lack of primal-dual range analysis symmetry. This illustrates the problem caused by primal degeneracy when doing objective function range analysis. If the primal is not degenerate there are not alternative dual optima. If the primal is degenerate there may be alternate dual optima and which dual optimal basis is picked affects the allowable increase and decrease on objective function coefficients.

To summarize, when examining solution printout for  $\min \{c^T x \mid Ax \geq b, x \geq 0\}$ , there are two ways to identify a solution that is not strictly complementary. Either there will be a primal variable with zero solution value and zero reduced cost, or an inequality constraint  $a_i^T x \geq b_i$  with zero slack and zero dual variable value. If the optimal solution is not strictly complementary, then there are alternative primal optima or alternative dual optimal. If there are alternative dual optima then every optimal primal solution is degenerate. If there are alternative primal optima then every optimal dual solution is degenerate.

### 3.5 CONCLUSION

Like projection, inverse projection is not practical for solving large linear programs. However, the importance of projection in proving theorems such as the finite basis theorem cannot be over emphasized. Later, in Chapters 11 and 16

we show how column generation, which is based on inverse projection, is a crucial tool for solving large, realistic problems. In the next chapter we extend the ideas of projection and inverse projection to inequalities with integer variables.

### 3.6 HOMEWORK EXERCISES

- 3.1 Find the extreme points and extreme rays of the recession cone of the polyhedron in  $\mathbb{R}^2$  defined by the inequalities

$$\begin{array}{rcl} x_2 & \geq & 1 \\ x_1 + x_2 & \geq & 2 \\ -0.5x_1 + x_2 & \leq & 8 \\ -x_1 + x_2 & \leq & 6 \\ x_1, x_2 & \geq & 0. \end{array}$$

- 3.2 Take the linear program<sup>1</sup>

$$\begin{array}{lll} \max & 10x_1 + 9x_2 & \\ & (7/10)x_1 + x_2 & \leq 630 \\ & (1/2)x_1 + (5/6)x_2 & \leq 600 \\ & x_1 + (2/3)x_2 & \leq 708 \\ & (1/10)x_1 + (1/4)x_2 & \leq 135 \\ & x_1, x_2 & \geq 0 \end{array}$$

and solve this linear program using inverse projection.

- 3.3 Assume  $A$  is an  $m \times n$  integer matrix with the property that in every column there is *at most* one strictly positive element and that the positive element is 1.0. There can be any number of integer negative nonzero elements in a column. Such a matrix is called *pre-Leontief*. Consider the polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b, x \geq 0\}$ . Prove that if  $b \geq 0$ , then all the extreme points of  $P$  are integer.
- 3.4 Assume that the system  $Ax \geq b, x \geq 0$  is *infeasible*. How would the inverse projection process detect this?
- 3.5 Take the linear program in Exercise 3.2 and calculate the allowable increase and decrease on both objective function coefficients.

---

<sup>1</sup>This is the Par Inc. problem from Anderson, Sweeney and Williams [12]

3.6 Construct an example of a minimization linear program which has a unique optimal dual solution, but the allowable decrease on the objective function coefficient of a primal variable  $\bar{x}_j = 0$ , is not equal to its reduced cost.

3.7 Prove Lemma 3.9.

3.8 Prove Proposition 3.10.

3.9 Construct an example of a minimization linear program which has alternative dual optima and decreasing the objective function coefficient of a variable  $x_j$  by  $c_j - \bar{u}A_{\bullet j}$ , where  $\bar{u}$  is an optimal dual solution, *will not* produce an alternate optimal solution with variable  $x_j$  at a positive value.

3.10 Prove that if there are alternative primal solutions to  $\min \{c^T x \mid Ax \geq b\}$ , then every optimal dual solution  $\bar{u}$  to the dual problem

$$\max \{b^T u \mid A^T u = c, u \geq 0\}$$

is degenerate. By degeneracy of a dual solution  $\bar{u}$  we mean that the rank of the matrix defined by the constraints  $A^T \bar{u} = c$  plus the nonnegativity constraints for which  $\bar{u}_i = 0$  is not full row rank.

3.11 Prove that the optimal value function

$$\theta(c) = \min \{c^T x \mid Ax \geq b\}$$

is piecewise linear and concave.

3.12 Extend the finite basis theorem for polyhedra to polyhedra of the form  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  (we are not assuming nonnegativity constraints  $x \geq 0$  as in Theorem 3.3).

3.13 Assume that there exists an optimal solution to  $\min \{c^T x \mid Ax \geq b\}$ . Prove that there exists a primal-dual solution  $(\bar{x}, \bar{u})$  which is not strictly complementary if and only if *for every* optimal primal-dual solution  $(\hat{x}, \hat{u})$  either  $\hat{x}$  is primal degenerate, or  $\hat{u}$  is dual degenerate.

3.14 *The 100% Rule.* Show that if the sum of the percentage increase (decrease) (that is the increase (decrease) in the objective function coefficient divided by the allowable increase (decrease) times 100) of all the objective function coefficients in the linear program,  $\min\{c^T x \mid Ax \geq b, x \geq 0\}$ , does not exceed 100% then the current optimal primal solution remains optimal.

---

# INTEGER LINEAR SYSTEMS: PROJECTION AND INVERSE PROJECTION

## 4.1 INTRODUCTION

In this chapter we address the problem of finding an  $x \in \mathbb{R}^n$  which is a solution to the system  $Ax = b$  or  $Ax \geq b$  with some or all of the  $x_i$  required to be integer. This problem has a very long history. Integer equalities were studied by the Greek mathematician Diophantos during the third century A.D. In fact, when all the variables are required to be integer the equations in the system  $Ax = b$  are called linear Diophantine equations. We use the terms Diophantine and integer interchangeably. The study of linear and nonlinear Diophantine equations is an important part of number theory. Although linear Diophantine equations were studied in the third century A.D. it was not until the 20th century A.D. (1976 to be precise) that a polynomial algorithm was given for finding an integer solution the system  $Ax = b$ .

A logical approach to finding an integer solution to  $Ax = b$  is to use the projection, or Gaussian elimination method introduced in Chapter 2. Consider the following example.

### Example 4.1

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 2 \\ 4x_1 + 3x_2 + x_3 &= 7 \end{aligned}$$

*Projecting out  $x_1$  gives the system*

$$x_2 - x_3 = 3$$

*An integer solution to this system is  $x_2 = 3, x_3 = 0$ . However, this corresponds to  $x_1 = -1/2$  which is not an integer.*

Unfortunately, an integer solution of the projected system may not correspond to an integer solution in the higher dimensional space. More machinery is required in order to use the projection philosophy for solving  $Ax = b$ , or  $Ax \leq b$  with  $x$  integer.

Closely related to the problem of finding integer solutions to  $Ax = b$  or  $Ax \geq b$  are the finite basis theorems. In Chapter 2 we showed using projection that the finitely generated sets

$$\begin{aligned} C &= \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q z_i x^i, z_i \geq 0, i = 1, \dots, q\}, \\ P &= \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i, \sum_{i=1}^q z_i = 1, z_i \geq 0, i = 1, \dots, r\} \end{aligned}$$

were a polyhedral cone and polyhedron, respectively. In Chapter 3 we used inverse projection to show the dual result that the cone  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$  and polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  were finitely generated. In this chapter, we investigate similar projection-inverse projection duality results when  $x$  is also required to be an integer vector.

In Section 4.2 we provide background material on integer monoids, lattices, linear congruences, the Euclidean algorithm and the Hermite normal form of a matrix. In Section 4.3 we give two polynomial algorithms for solving systems of linear congruences. One algorithm is based on projection, the other is based on inverse projection using the Hermite normal form of a matrix. Two polynomial algorithms for solving  $Ax = b$ ,  $x$  integer are given in Section 4.4. These algorithms are based on the corresponding projection and inverse projection algorithms for solving linear congruences developed in the previous section. In Section 4.5 a projection algorithm is given for finding a solution to  $Ax \geq b$ ,  $x$  integer. In Section 4.6 we give an inverse projection algorithm for finding a solution to  $Ax \geq b$ ,  $x$  integer and provide several finite basis theorems. Concluding remarks are given in Section 4.7. Exercises are provided in Section 4.8.

## 4.2 BACKGROUND MATERIAL

### 4.2.1 Lattices, Monoids, Unimodular Matrices and Congruences

Let  $\mathcal{Z}$  denote the set of integers and  $\mathcal{Z}_+$  the set of nonnegative integers. If  $b^i$  for  $i = 1, \dots, q$  are rational  $n$ -vectors, then the set of vectors  $G = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q z_i b^i, z_i \in \mathcal{Z}, i = 1, \dots, q\}$  is a *lattice* in  $\mathbb{R}^n$  generated by  $b^1, \dots, b^q$ . The rational vectors  $b^1, \dots, b^q$  are a *basis* of the lattice. A lattice is an (additive) group  $G$ , in particular,  $0 \in G$ , if  $x \in G$  then  $-x \in G$  and  $x, y \in G$  implies  $x + y \in G$ . It is of interest to drop the requirement that if  $x$  is in the set, its inverse element  $-x$  is also in the set.

A *monoid*  $M \subset \mathbb{R}^n$  is a semi-group of real vectors which also contains the zero vector. In particular,  $0 \in M$  and  $x, y \in M$  implies  $x + y \in M$ . In this chapter we restrict our attention to integer monoids, that is  $M \subseteq \mathcal{Z}^n$ . Of particular interest are the *polyhedral monoids* defined by

$$M = \{x \in \mathcal{Z}^n \mid Ax \geq 0\}.$$

In Section 4.6 we show using inverse projection that polyhedral monoids are finitely generated. The converse result that a finitely generated monoid is a polyhedral monoid does not hold.

A nonsingular matrix  $A$  is *unimodular* if it is an integer matrix and has determinant  $\pm 1$ . Later, when solving  $Ax = b$  it is necessary to perform column (row) operations on  $A$  which preserve the unimodularity of  $A$ . It follows from properties of determinants that the following operations preserve the unimodularity of a matrix.

#### Unimodular Column Operations

1. exchanging two columns
2. multiplying a column by -1
3. adding an integer multiple of one column to another

#### Unimodular Row Operations:

1. exchanging two rows

2. multiplying a row by -1
3. adding an integer multiple of one row to another

Let  $\theta$  be a positive integer. If  $a, b \in \mathbb{Z}$ , then  $a$  is congruent to  $b$  modulo  $\theta$  if  $(a - b)/\theta$  is an integer. This is written as

$$a \equiv b \pmod{\theta}.$$

In Section 4.3 we give a polynomial algorithms for finding integer solutions to systems of congruences.

### 4.2.2 The Euclidean Algorithm

Consider the simplest interesting version of  $Ax = b$ ,  $x$  integer, namely  $a_1x_1 + a_2x_2 = b$  where both  $x_1$  and  $x_2$  are integer variables. Without loss, assume  $a_1, a_2$  and  $b$  are integers. Let  $\gcd(a_1, a_2)$  denote the greatest common divisor (gcd) of  $a_1$  and  $a_2$ . For example,  $\gcd(12, 15) = 3$ . Finding  $\gcd(a_1, a_2)$  is the key to solving the Diophantine equation  $a_1x_1 + a_2x_2 = b$ . One could try to find the gcd by trial and error but that is obviously not efficient. The gcd is found by the famous Euclidean algorithm and it is based on the following lemma.

**Lemma 4.2** *Let  $a_1, a_2$  be integers. If  $a_1 = qa_2 + r$  with  $q, r$  integer, then  $\gcd(a_1, a_2) = \gcd(a_2, r)$ .*

Assume without loss, that  $a_1$  and  $a_2$  are positive. The gcd of  $a_1, a_2$  is found by recursively applying Lemma 4.2. If  $a_1 \geq a_2 > 0$ , calculate

$$q = \lfloor a_1/a_2 \rfloor \quad r = a_1 - qa_2 \tag{4.1}$$

$$a_1 = a_2 \quad a_2 = r \tag{4.2}$$

The largest integer less than or equal to  $a$  is denoted by  $\lfloor a \rfloor$ . By recursively applying (4.1)-(4.2) we eventually get an integer pair  $a_1, a_2$  where  $a_2 = 0$ . Assume for input integers  $a_1 \geq a_2 > 0$ . The Euclidean algorithm is

#### Algorithm 4.3 (Euclidean Algorithm)

```
while( a1 > 0 and a2 > 0 ) {
    q ← ⌊ a1/a2 ⌋
```

```

 $r \leftarrow a_1 - qa_2$ 
 $a_1 \leftarrow a_2$ 
 $a_2 \leftarrow r$ 
}

```

The notation  $a_1 \leftarrow a_2$  means replace the memory location of  $a_1$  with the quantity in the location of  $a_2$ . Throughout the text, in formal statements of algorithms, we replace  $=$  with  $\leftarrow$  if the left hand side is replaced or updated by the right hand side.

It follows from Lemma 4.2 that the final value of  $a_1$  in Algorithm 4.3 is the desired gcd. It is left as an exercise to prove that the Euclidean algorithm has complexity  $O(\max\{\log_2(a_1), \log_2(a_2)\})$ .

**Example 4.4** Let  $a_1 = 10200, a_2 = 3553$ . Applying the Euclidean algorithm gives:

*Iteration 1:  $q = \lfloor 10200/3553 \rfloor = 2$  and  $r = 10200 - 2 * 3553 = 3094$ . Then the new pair is  $a_1 = 3553, a_2 = 3094$ .*

*Iteration 2:  $q = \lfloor 3553/3094 \rfloor = 1$  and  $r = 3553 - 3094 = 459$ . The new pair is  $a_1 = 3094, a_2 = 459$ .*

*Iteration 3:  $q = \lfloor 3094/459 \rfloor = 6$  and  $r = 3094 - 6 * 459 = 340$ . The new pair is  $a_1 = 459, a_2 = 340$ .*

*Iteration 4:  $q = \lfloor 459/340 \rfloor = 1$  and  $r = 459 - 340 = 119$ . The new pair is  $a_1 = 340, a_2 = 119$ .*

*Iteration 5:  $q = \lfloor 340/119 \rfloor = 2$  and  $r = 340 - 2 * 119 = 102$ . The new pair is  $a_1 = 119, a_2 = 102$ .*

*Iteration 6:  $q = \lfloor 119/102 \rfloor = 2$  and  $r = 119 - 2 * 102 = 17$ . The new pair is  $a_1 = 102, a_2 = 17$ .*

*Iteration 7:  $q = \lfloor 102/17 \rfloor = 6$  and  $r = 102 - 6 * 17 = 0$ . The new pair is  $a_1 = 17, a_2 = 0$ . We are done, gcd = 17. Repeated application of Lemma 4.2 gives*

$$\begin{aligned}
 \gcd(10200, 3553) &= \gcd(3553, 3094) = \gcd(3094, 459) = \gcd(459, 340) \\
 &= \gcd(340, 119) = \gcd(119, 102) = \gcd(102, 17) \\
 &= \gcd(17, 0) = 17.
 \end{aligned}$$

The gcd is used to find a solution to a single Diophantine equation with two variables.

**Proposition 4.5** *There is an integer solution to the system  $a_1x_1 + a_2x_2 = b$  if and only if  $\gcd(a_1, a_2)$  divides  $b$ .*

**Proof:** Assume there is an integer solution  $\bar{x}_1, \bar{x}_2$  to the system  $a_1x_1 + a_2x_2 = b$ . By definition  $\gcd(a_1, a_2)$  divides  $a_1$  and  $a_2$ . Then  $\gcd(a_1, a_2)$  divides both  $a_1\bar{x}_1$  and  $a_2\bar{x}_2$ . Then  $\gcd(a_1, a_2)$  divides  $b = a_1\bar{x}_1 + a_2\bar{x}_2$ .

The converse of this proposition is by construction and the process is outlined below.  $\square$

Construct an integer solution to  $a_1x_1 + a_2x_2 = b$  by first solving the equation  $a_1x_1 + a_2x_2 = \gcd(a_1, a_2)$ . The Euclidean algorithm provides a solution to this system.

**Example 4.6** Refer back to Example 4.4. At each step of the Euclidean algorithm we had the following.

$$\begin{aligned} 17 &= 119 - 1 * 102 \\ 102 &= 340 - 2 * 119 \\ 119 &= 459 - 1 * 340 \\ 340 &= 3094 - 6 * 459 \\ 459 &= 3553 - 1 * 3094 \\ 3094 &= 10200 - 2 * 3553 \end{aligned}$$

Observe that  $\gcd(10200, 3553) = 17$  and  $x_1 = 1, x_2 = -1$  is a solution to the system  $119x_1 + 102x_2 = 17$ . But  $x_1 = 1, x_2 = -2$  is a solution to  $340x_1 + 119x_2 = 102$ . Thus,

$$17 = 119 - 1(102) = 119 - 1 * (340 - 2 * 119) = 3 * 119 - 1 * 340$$

and  $x_1 = 3, x_2 = -1$  is a solution to  $119x_1 + 340x_2 = 17$ . Working backward in this fashion we get a solution to  $10200x_1 + 3553x_2 = 17$ . A solution is  $x_1 = -31, x_2 = 89$ .

This solution to  $a_1x_1 + a_2x_2 = b$  is found in general by observing that the Euclidean algorithm corresponds unimodular column operations on the matrix

$$E = \begin{bmatrix} a_1 & a_2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

In the iterative step of the Euclidean algorithm we multiply the second column by  $-\lfloor a_1/a_2 \rfloor$ , add it to the first column and then exchange the first and second columns. Initially,

$$[0, 0] = [-1, a_1, a_2]E. \quad (4.3)$$

Since zero plus an integer times zero is zero, the column operations performed on  $E$  do not affect this equality. Therefore, when the Euclidean algorithm terminates

$$a_1t_{11} + a_2t_{21} = \gcd(a_1, a_2) \quad (4.4)$$

where the  $t_{ij}$  are the elements in row  $i+1$  and column  $j$  of matrix  $E$  upon termination of the Euclidean algorithm. Then  $x_1 = t_{11}$ ,  $x_2 = t_{21}$  is a solution to the Diophantine equation  $a_1x_1 + a_2x_2 = \gcd(a_1, a_2)$ . If  $\gcd(a_1, a_2)$  divides  $b$ , then a solution to  $a_1x_1 + a_2x_2 = b$  is found by multiplying any solution to  $a_1x_1 + a_2x_2 = \gcd(a_1, a_2)$  by  $b/\gcd(a_1, a_2)$ .

Actually, a whole family of integer solutions can be generated to (4.4). From (4.3) it follows that  $a_1t_{12} + a_2t_{22} = 0$ . Then all solutions to the Diophantine equation  $a_1x_1 + a_2x_2 = b$  are characterized by Proposition 4.7 below.

**Proposition 4.7** *If  $\gcd(a_1, a_2)$  divides  $b$  then every integer solution to  $a_1x_1 + a_2x_2 = b$  is given by*

$$\begin{aligned} x_1 &= t_{11}(b/\gcd(a_1, a_2)) + st_{12} \\ x_2 &= t_{21}(b/\gcd(a_1, a_2)) + st_{22} \end{aligned}$$

for some integer  $s$ .

There is also a very simple, nonconstructive proof for Proposition 4.5 using basic results about ideals defined on the ring of integers. See, for example, Armendáriz and McAdam [16].

### 4.2.3 Inverse Projection and The Euclidean Algorithm

We now view the Euclidean algorithm as an example of inverse projection. Although the ancient Greeks didn't think about inverse projection, there is a nice interpretation of the Euclidean algorithm as an example of inverse projection. This is the first example in this book of using inverse projection to solve a problem with integer variables. Throughout the book, inverse projection plays a key role in solving integer programming problems. Later, in Chapter 16 we show how inverse projection is used to solve large scale, real world problems.

The Euclidean algorithm involves column operations. In general, one can think of column operations as inverse projection in the sense that column operations correspond to a variable transformation. The variable transformation introduces new or *auxiliary variables*. The Euclidean algorithm is used to solve the equation

$$a_1x_1 + a_2x_2 = b \quad (4.5)$$

by applying unimodular column operations to the matrix

$$E = \begin{bmatrix} a_1 & a_2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

When the Euclidean algorithm terminates, the matrix  $E$  is transformed into the new matrix

$$\bar{E} = \begin{bmatrix} \gcd(a_1, a_2) & 0 \\ t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}.$$

Denote by  $T$  the matrix obtained from  $\bar{E}$  by deleting the first row. A new system using *auxiliary variables* equivalent to (4.5) is

$$[a_1 \ a_2]Tz = b \quad (4.6)$$

$$x = Tz. \quad (4.7)$$

**Example 4.8 ( Example 4.4 Continued)** The Diophantine equation is  $10200x_1 + 3553x_2 = 85$ . Upon conclusion of the Euclidean algorithm

$$\bar{E} = \begin{bmatrix} 17 & 0 \\ -31 & 209 \\ 89 & -600 \end{bmatrix}.$$

The implied variable transformation matrix is

$$T = \begin{bmatrix} -31 & 209 \\ 89 & -600 \end{bmatrix}.$$

Thus,  $x_1 = -31z_1 + 209z_2$  and  $x_2 = 89z_1 - 600z_2$ . Substituting into  $10200x_1 + 3553x_2 = 85$  gives  $17z_1 + 0z_2 = 85$ . In particular we have the following. Let

$$\Gamma = \{(x_1, x_2) \mid 10200x_1 + 3553x_2 = 85, x_1, x_2 \text{ integer}\}$$

and

$$17z_1 + 0z_2 = 85 \quad (4.8)$$

$$-31z_1 + 209z_2 = x_1 \quad (4.9)$$

$$89z_1 - 600z_2 = x_2. \quad (4.10)$$

Observe that  $\bar{x} \in \Gamma$  if and only if there is a corresponding  $\bar{z}$  such that  $(\bar{x}, \bar{z})$  is a solution to (4.8)-(4.10). Of course (4.8)-(4.10) is trivial to solve. Find a solution to (4.8) and then use (4.9)-(4.10) to give the corresponding  $\bar{x}$ .

We generalize this idea in the next section with the introduction of the Hermite normal form for a matrix and then use the auxiliary variable method to develop an algorithm for solving systems of congruences. This algorithm is then used to find an integer solution to the system  $Ax = b$ .

#### 4.2.4 Hermite Normal Form

The inverse projection idea based on the Euclidean algorithm for solving a two variable Diophantine is easily extended to solve the system  $Ax = b$ . We find a set of auxiliary variables  $z$  and a linear transformation  $x = Tz$  such that the new “equivalent” system is  $ATz = b$ ,  $z$  integer. There are two properties we would like the matrix  $T$  to have. First, we want the system  $ATz = b$  to be trivial to solve in the new variables – why do a transformation if we don’t make the problem easier to solve. Second, we want the two systems equivalent so there is a one to one correspondence between integer solutions to  $Ax = b$  and  $ATz = b$ . If both of these conditions are met then an integer solution to  $Ax = b$  is found by finding an integer solution to the easier system  $ATz = b$  and then making the transformation  $x = Tz$ .

If  $AT = [B \ 0]$  where  $B$  is a lower triangular matrix then the first condition is satisfied since an integer solution is found (or we show one does not exist)

using forward substitution. If  $T$  and  $T^{-1}$  are integer matrices then the second condition is satisfied since if  $x$  integer then  $z = T^{-1}x$  is integer and if  $z$  is integer then  $x = Tz$  is integer. In the two variable case, these conditions were easily met since  $ATz$  became  $\gcd(a_1, a_2)z_1 + 0z_2 = b$  and  $T$  is a unimodular matrix, i.e. is integer and has determinant either +1 or -1 which implies  $T^{-1}$  is integer.

In the general case, the  $T$  matrix is constructed so that  $AT$  is in Hermite normal form. A matrix of full row rank  $E$  is in *Hermite normal form* if

1.  $E = [B \ 0]$ ;
2. the matrix  $B$  is a nonsingular, lower triangular, with positive diagonal elements and nonpositive off-diagonal elements;
3. each diagonal element of  $B$  is the unique maximum (after taking absolute values) entry in that row, i.e.  $b_{ii} > |b_{ij}|$  for  $j = 1, \dots, i-1$ .

Any integer matrix of full row rank is brought into Hermite normal form by using the unimodular column operations of exchanging two columns, multiplying a column by -1 and adding an integer multiple of one column to another. The iterative step of the algorithm to put an integer matrix  $A$  into Hermite normal form proceeds as follows. Assume at an intermediate iteration the matrix has the form

$$\begin{bmatrix} B & 0 \\ C & D \end{bmatrix}.$$

At this iteration, the submatrix  $B$  is in Hermite normal form. First make every element in the first row of  $D$  nonnegative by multiplying any column with a negative element by -1. Then apply the Euclidean algorithm to each pair of elements in the first row of matrix  $D$ . That is, if there exist two positive elements  $d_{1j} > d_{1k}$  add  $-\lfloor d_{1j}/d_{1k} \rfloor$  times column  $k$  to column  $j$ . Repeating this process and making the necessary column exchanges gives  $d_{11} > 0$  and all other  $d_{1j} = 0$ . Repeated application of this step results in a matrix  $[B \ 0]$  where  $B$  is lower triangular and the diagonal elements of  $B$  are positive. Proceeding in row order, it is now trivial to make satisfy the other conditions of Hermite normal form by adding  $-\lceil b_{ij}/b_{ii} \rceil$  for each  $j < i$ . The smallest integer greater than or equal to  $a$  is denoted by  $\lceil a \rceil$ .

#### Example 4.9

$$\begin{bmatrix} 340 & 119 & 18 \\ 4 & 12 & -6 \end{bmatrix}$$

Perform the Euclidean algorithm on the pair 340 and 119 and find a gcd of 17. The updated matrix is

$$\begin{bmatrix} 0 & 17 & 18 \\ -212 & 32 & -6 \end{bmatrix}.$$

The Euclidean algorithm applied to the pair 17 and 18 gives the gcd of 1 and the new matrix

$$\begin{bmatrix} 0 & 0 & 1 \\ -212 & 678 & -38 \end{bmatrix}.$$

Exchange the first and third column and apply the Euclidean algorithm to the pair -212 and 678. This gives the lower triangular matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ -38 & 2 & 0 \end{bmatrix}.$$

Adding 14 times column two to column one gives the Hermite normal form of the matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

Once  $A$  has been put into Hermite normal form observe that the system  $Ax = b$  has a solution if and only if  $B^{-1}b$  is an integer matrix. This gives the following important theorem of the alternative.

**Proposition 4.10** If  $A, b$  are integer, either the system

$$Ax = b, \quad x \text{ integer} \tag{4.11}$$

has a solution, or the system

$$A^T u \text{ integer}, \quad b^T u \text{ not integer} \tag{4.12}$$

has a solution, but not both.

**Proof:** Assume  $Ax = b$  has an integer solution. If  $\bar{x}$  is an integer solution to  $Ax = b$  and  $\bar{u}^T A$  is an integer vector, then  $\bar{u}^T b = \bar{u}^T A \bar{x}$  must be an integer vector and (4.12) cannot have a solution. Therefore, both systems cannot have a solution.

Now show that at least one system holds. Assume without loss  $A$  is of full row rank and  $[B \ 0]$  is the Hermite normal form of  $A$ . If  $B^{-1}b$  is an integer vector then system (4.11) has a solution. If  $B^{-1}b$  is not an integer vector, there is a

component  $i$  not an integer. Since  $B^{-1}B = I$  the product of the  $i$ th row of  $B^{-1}$  with  $B$  is an integer vector. Let  $u$  be the  $i$ th row of  $B^{-1}$  and the alternative condition  $b^T u$  not an integer is satisfied.  $\square$

What is the complexity of putting an  $m \times n$  integer matrix  $A$ , into Hermite normal form? The basic operation is to take a pair of integers and find their gcd. Thus, the Euclidean algorithm is called at most  $O(mn)$  times and the Euclidean algorithm is a polynomial algorithm. However, there is a problem. The numbers in the *intermediate matrices* may get very large and require an amount of space which is not a polynomial function of the binary encoding of the data in  $A$ . It is possible to keep the numbers from growing too large. Clearly, a polynomial algorithm for the Hermite normal form of a matrix implies the existence of a polynomial algorithm for determining if there is an integer solution to the system  $Ax = b$ . See, for example, Frumkin [162, 163], Kannan and Bachem [257], von zur Gathen and Sieveking [169], and Domich, Kannan, and Trotter [128]. In Section 4.4, we use the Hermite normal form for an inverse projection algorithm for solving linear Diophantine equations. In that section we show it is possible to bound the size of the numbers by the largest determinant of a submatrix of  $A$ .

## 4.3 SOLVING A SYSTEM OF CONGRUENCE EQUATIONS

Here we show how to solve a system of  $m$  congruences in  $n$  variables in polynomial time. We show how this can be done using projection and then inverse projection. First consider finding all solutions to a single congruence.

### 4.3.1 Solving One Congruence Using Projection

In this subsection we show how determine all the solutions to a congruence relation by successively projecting out variables<sup>1</sup>. Consider

$$a_{11}x_1 + \cdots + a_{1n}x_n \equiv b_1 \pmod{\theta}$$

where  $a_{11}, \dots, a_{1n}$  and  $b_1$  are integers and  $\theta$  is a positive integer. Just as with Gaussian elimination, the key projection idea is to project out a variable and

---

<sup>1</sup>Many of the ideas in this section were developed jointly with G. Purdy at the University of Cincinnati during the Spring of 1992.

replace the  $n$  variable problem with an “equivalent”  $n - 1$  variable problem. The equivalent problem we use is based on Lemma 4.11.

**Lemma 4.11 (Congruence Projection)** *If  $\alpha = \gcd(a_{11}, \theta)$  then  $(\bar{x}_2, \dots, \bar{x}_n)$  is a solution to*

$$a_{12}x_2 + \cdots + a_{1n}x_n \equiv b_1 \pmod{\alpha} \quad (4.13)$$

*if and only if there exists  $\bar{x}_1$  such that  $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$  is a solution to*

$$a_{11}x_1 + \cdots + a_{1n}x_n \equiv b_1 \pmod{\theta}. \quad (4.14)$$

**Proof:** If there exists  $\bar{x}_1$  such that  $(\bar{x}_1, \dots, \bar{x}_n)$  is a solution to (4.14) then  $\theta$  divides  $(\sum_{i=1}^n a_{1i}\bar{x}_i - b)$ . Since  $\alpha$  is the gcd of  $a_{11}$  and  $\theta$  it follows that  $\alpha$  also divides  $(a_{11}\bar{x}_1 + \sum_{i=2}^n a_{1i}\bar{x}_i - b)$ . But  $\alpha$  divides  $a_{11}$  so it divides  $a_{11}\bar{x}_{11}$ . Then  $\alpha$  must also divide  $(\sum_{i=2}^n a_{1i}\bar{x}_i - b)$ .

Next assume that  $(\bar{x}_2, \dots, \bar{x}_n)$  is a solution to (4.13). Then there exists  $\bar{x}_1$  such that  $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$  is a solution to (4.14) if there is a solution to

$$a_{11}x_1 \equiv b_1 - a_{12}\bar{x}_2 - \cdots - a_{1n}\bar{x}_n \pmod{\theta}. \quad (4.15)$$

The congruence given by (4.15) is equivalent to the two variable Diophantine equation

$$a_{11}x_1 - \theta x_0 = (b - a_{12}\bar{x}_2 - \cdots - a_{1n}\bar{x}_n). \quad (4.16)$$

By construction,  $\alpha$  is the gcd of  $a_{11}$  and  $\theta$  and  $\alpha$  divides

$$(b - a_{12}\bar{x}_2 - \cdots - a_{1n}\bar{x}_n).$$

Then the Diophantine equation given by (4.16) has an integer solution.  $\square$

This process is repeated until one of the  $a_{1j}$  coefficients is prime to the modulus. The resulting congruence is then trivial to solve.

**Example 4.12** Assume the congruence relation is

$$882x_1 + 14x_2 + 4x_3 + 5x_4 + 9x_5 \equiv 15 \pmod{2205}. \quad (4.17)$$

Since  $441 = \gcd(882, 2205)$  replace (4.17) with

$$14x_2 + 4x_3 + 5x_4 + 9x_5 \equiv 15 \pmod{441}. \quad (4.18)$$

Since  $7 = \gcd(14, 441)$  project out variable  $x_2$  and solve the congruence

$$4x_3 + 5x_4 + 9x_5 \equiv 15 \pmod{7}.$$

Reducing ( $\bmod 7$ ) gives

$$4x_3 + 5x_4 + 2x_5 \equiv 1 \pmod{7}.$$

The integers 4 and 7 are relatively prime, so we stop the process and solve

$$4x_3 \equiv (1 - 5x_4 - 2x_5) \pmod{7}.$$

This is equivalent to the Diophantine equation

$$4x_3 - 7x_0 = (1 - 5x_4 - 2x_5).$$

Applying the generalized Euclidean algorithm to this Diophantine equation transforms through unimodular column operations the matrix

$$\left[ \begin{array}{cc} 4 & -7 \\ 1 & 0 \\ 0 & 1 \end{array} \right]$$

into the matrix

$$\left[ \begin{array}{cc} 1 & 0 \\ 2 & 7 \\ 1 & 4 \end{array} \right].$$

Applying Proposition 4.7 with  $b = 1 - 5x_4 - 2x_5$  and  $\gcd(4, -7) = 1$  gives:

$$x_3 = (2/1)(1 - 5x_4 - 2x_5) + 7s_3 = 2 - 10x_4 - 4x_5 + 7s_3.$$

Substitute this solution back into equation (4.18) and get

$$14x_2 + 28s_3 - 35x_4 - 7x_5 \equiv 7 \pmod{441}. \quad (4.19)$$

Next solve,

$$14x_2 \equiv (7 - 28s_3 + 35x_4 + 7x_5) \pmod{441}.$$

Solve this by finding a solution to the Diophantine equation

$$14x_2 - 441x_0 = (7 - 28s_3 + 35x_4 + 7x_5).$$

Applying the generalized Euclidean algorithm to this Diophantine equation transforms through column operations the matrix

$$\left[ \begin{array}{cc} 14 & -441 \\ 1 & 0 \\ 0 & 1 \end{array} \right]$$

into the matrix

$$\begin{bmatrix} 0 & 7 \\ 63 & -31 \\ 2 & -1 \end{bmatrix}.$$

Applying Proposition (4.7) with  $b = (7 - 28s_3 + 35x_4 + 7x_5)$  and  $\gcd(14, -441) = 7$  gives

$$x_2 = (-31/7)(7 - 28s_3 + 35x_4 + 7x_5) + 63s_2 = -31 + 124s_3 - 155x_4 - 31x_5 + 63s_2.$$

Substituting these solutions for  $x_3$  and  $x_2$  back into equation (4.17) gives

$$882x_1 + 882s_2 + 1764s_3 - 2205x_4 - 441x_5 \equiv 441 \pmod{2205}$$

Reducing (mod 2205) gives

$$882x_1 + 882s_2 + 1764s_3 - 0x_4 - 441x_5 \equiv 441 \pmod{2205}$$

Finally, solve the Diophantine equation

$$882x_1 - 2205x_0 = (441 - 882s_2 - 1764s_3 + 441x_5).$$

Applying the generalized Euclidean algorithm to this Diophantine equation transforms through column operations the matrix

$$\begin{bmatrix} 882 & -2205 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

into the matrix

$$\begin{bmatrix} 0 & 441 \\ 5 & -2 \\ 2 & -1 \end{bmatrix}.$$

Applying Proposition (4.7) with  $b = (441 - 882s_2 - 1764s_3 + 441x_5)$  and  $\gcd(882, -2205) = 441$  gives

$$x_1 = (-2/441)(441 - 882s_2 - 1764s_3 + 441x_5) + 5s_1 = -2 + 4s_2 + 8s_3 - 2x_5 + 5s_1.$$

Variables  $x_4$  and  $x_5$  are free to set at any value, say  $s_4$  and  $s_5$ , respectively. The complete solution set for this problem is

$$\begin{array}{rcl} x_1 & = & -2 + 5s_1 + 4s_2 + 8s_3 + 0s_4 - 2s_5 \\ x_2 & = & -31 + 63s_2 + 124s_3 - 155s_4 - 31s_5 \\ x_3 & = & 2 + 7s_3 - 10s_4 - 4s_5 \\ x_4 & = & 0 + s_4 \\ x_5 & = & 0 + s_5. \end{array} \quad (4.20)$$

Letting  $s_1, s_2, s_3, s_4$ , and  $s_5$  range over the integers generates the solution set to (4.17). This solution set is characterized by a vector, plus a lattice generated by  $n$  linearly independent vectors.

In general, for  $\theta > 0$ , we need to execute the projection step of this algorithm at most  $\lceil \log_2(\theta) \rceil$  times since at each time a variable is projected the modulus  $\theta$  gets factored. Each projection step requires two gcd calculations (factoring the modulus and solving the underlying two variable Diophantine equation). Thus, the amount of work in the gcd calculations is  $O((\log_2(\theta))^2)$  since none of the coefficients in the congruence ever exceed  $\theta$ . Further, if there are  $n$  variables then substituting the variable just solved for back into the previous equation requires  $O(n)$  multiplications and additions. Thus, finding all solutions for one congruence requires work  $O((n + \log_2(\theta)) \log_2(\theta))$ .

### 4.3.2 Solving One Congruence by Inverse Projection

We now show how to find all solutions to a congruence relation using inverse projection. Consider the congruence relation

$$a_{11}x_1 + \cdots + a_{1n}x_n \equiv b_1 \pmod{\theta}. \quad (4.21)$$

Define the matrix

$$E = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \quad (4.22)$$

Perform unimodular column operations on  $E$  so that the first row of  $E$  is in Hermite normal form. Let  $\bar{E}$  be the resulting matrix and let  $T$  denote the  $n \times n$  matrix obtained by deleting the first row of  $\bar{E}$ . The column operations used to put a matrix in Hermite normal form are unimodular operations so  $T$  and  $T^{-1}$  are integer matrices. Use  $T$  to define a new system with auxiliary variables.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \end{bmatrix} Tz \equiv b_1 \pmod{\theta} \quad (4.23)$$

$$Tz = x \quad (4.24)$$

**Proposition 4.13** *The integer vector  $\bar{x}$  is a solution to (4.21) if and only if there is an integer vector  $\bar{z}$  such that  $(\bar{x}, \bar{z})$  is a solution to (4.23)-(4.24).*

**Proof:** The column operations used transform  $E$  into  $\bar{E}$  are unimodular column operations so the determinant of  $T$  is either +1 or -1. Therefore,  $T^{-1}$  is an integer matrix. Then given integer  $\bar{x}$ ,  $T^{-1}\bar{x}$  is integer and the result is immediate.  $\square$

Since the first row of  $\bar{E}$  is in Hermite normal form, there is one nonzero element in the first row, namely  $\bar{e}_{11} = \gcd(a_{11}, a_{12}, \dots, a_{1n})$ . Then (4.23) becomes

$$\bar{e}_{11}z_1 \equiv b_1 \pmod{\theta}. \quad (4.25)$$

with  $z_2, \dots, z_n$  free to set at any integer value. The congruence (4.25) has a solution if and only if  $\gcd(\bar{e}_{11}, \theta)$  divides  $b_1$ . If  $\gcd(\bar{e}_{11}, \theta)$  does divide  $b_1$ , then there exist  $\alpha_0, \alpha_1$  such that all solutions to (4.23)-(4.24) are given by

$$z_1 = \alpha_0 + s_1\alpha_1, \quad z_2 = s_2, \dots, z_n = s_n \quad (4.26)$$

where  $s_1, \dots, s_n \in \mathbb{Z}$ . All solutions in terms of the  $x$  variables are

$$x = T \begin{bmatrix} \alpha_0 + s_1\alpha_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}. \quad (4.27)$$

**Example 4.14 (Example 4.12 continued)** Solve the congruence

$$882x_1 + 14x_2 + 4x_3 + 5x_4 + 9x_5 \equiv 15 \pmod{2205}$$

using inverse projection.

$$E = \begin{bmatrix} 882 & 14 & 4 & 5 & 9 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Carry out the unimodular column operations necessary to put the first row of  $E$  in Hermite normal form. First perform the column operations to find  $\gcd(882, 14)$ .

$$\begin{bmatrix} 0 & 14 & 4 & 5 & 9 \\ 1 & 0 & 0 & 0 & 0 \\ -63 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Next perform the column operations to find  $\gcd(14, 4)$ .

$$\left[ \begin{array}{ccccc} 0 & 2 & 0 & 5 & 9 \\ 1 & 0 & 0 & 0 & 0 \\ -63 & 1 & -2 & 0 & 0 \\ 0 & -3 & 7 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right]$$

Now find  $\gcd(2, 5)$ . This requires multiplying the second column by  $-2$  and adding it to the fourth column.

$$\left[ \begin{array}{ccccc} 0 & 2 & 0 & 1 & 9 \\ 1 & 0 & 0 & 0 & 0 \\ -63 & 1 & -2 & -2 & 0 \\ 0 & -3 & 7 & 6 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right]$$

Now that there is a 1 in the fourth column, multiply the fourth column by  $-2$  and add it to the second column. Then multiply the fourth column by 9 and add it to the fifth column.

$$\left[ \begin{array}{ccccc} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ -63 & 5 & -2 & -2 & 18 \\ 0 & -15 & 7 & 6 & -54 \\ 0 & -2 & 0 & 1 & -9 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right]$$

Finally, exchange the first and fourth columns.

$$\left[ \begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -2 & 5 & -2 & -63 & 18 \\ 6 & -15 & 7 & 0 & -54 \\ 1 & -2 & 0 & 0 & -9 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right]$$

Drop the first row and define the transformation matrix

$$T = \left[ \begin{array}{ccccc} 0 & 0 & 0 & 1 & 0 \\ -2 & 5 & -2 & -63 & 18 \\ 6 & -15 & 7 & 0 & -54 \\ 1 & -2 & 0 & 0 & -9 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right]. \quad (4.28)$$

Making the substitution  $x = Tz$  in the congruence relation 4.17 gives

$$z_1 + 0z_2 + 0z_3 + 0z_4 + 0z_5 \equiv 15 \pmod{2205}$$

The solution set to this congruence relation is

$$z_1 = 15 + 2205s_1, \quad z_2 = s_2, \quad z_3 = s_3, \quad z_4 = s_4, \quad z_5 = s_5.$$

But  $x = Tz$ , so the solution set in the original  $x$  variables is

$$\begin{array}{rcl} x_1 & = & 0 \\ x_2 & = & -30 & -4410s_1 & +5s_2 & -2s_3 & -63s_4 & +18s_5 \\ x_3 & = & 90 & +13230s_1 & -15s_2 & +7s_3 & +0s_4 & -54s_5 \\ x_4 & = & 15 & +2205s_1 & -2s_2 & & & -9s_5 \\ x_5 & = & 0 & & & & & +s_5 \end{array} \quad (4.29)$$

Letting  $s_1, s_2, s_3, s_4$ , and  $s_5$  range over the set of integers generates the solution set to (4.17). Once again, this solution set is characterized by a vector, plus a lattice generated by  $n$  linearly independent vectors. We leave it as an exercise to show that the solution set generated by (4.29) is identical to the solution set (4.20) found using projection.

The major step in the inverse projection method for one congruence is to find the gcd of  $n$  integers and calculate the  $n \times n$  transformation matrix  $T$ . Then at each iteration of the Euclidean algorithm, an  $n$ -vector is multiplied by a scalar and added to another  $n$ -vector. There are at most  $O(n \log_2(\theta))$  iterations for a complexity of  $O(n^2 \log_2(\theta))$ .

### 4.3.3 More Than One Congruence

The projection and inverse projection methods for solving one congruence are easily extended to systems involving more than one congruence. Consider the projection method first. Using the method developed earlier, find the family of solutions to the first congruence and substitute these values into the remaining  $m - 1$  congruences. Finding all solutions to a single congruence  $i$  requires work  $O((n + \log_2(\theta_i)) \log_2(\theta_i))$  where  $\theta_i > 0$  is the modulus of congruence relation  $i$ . The family of solutions to congruence  $i$  must be substituted into the remaining congruences. There are  $O(\log_2(\theta_i))$  variables which are expressed as a linear combination of  $n$  new variables. The substitution must be made in  $O(m)$  constraints for work  $O(mn \log_2(\theta_i))$ . This must be done potentially  $m$  times for a total algorithm complexity of  $O(n^2 n \log_2(\theta_{max}) + m \log_2^2(\theta_{max}))$

operations where  $\theta_{max}$  is the maximum congruence modulus. The analysis for the inverse projection method is similar.

The following proposition follows from both the projection and inverse projection methods for solving congruence systems.

**Proposition 4.15** *If there is a feasible solution to the system*

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \equiv b_i \pmod{\theta_i}, \quad i = 1, \dots, m,$$

*with all  $a_{ij}$  and  $b_i$  rational, then there exist linearly independent vectors,  $x^1, \dots, x^n$  and a vector  $x^0$  such that  $x$  is a solution if and only if*

$$x = x^0 + G$$

*where  $G$  is the lattice*

$$G = \{y \mid y = z_1x^1 + z_2x^2 + \cdots + z_nx^n, z_1, z_2, \dots, z_n \in \mathbb{Z}\}.$$

## 4.4 INTEGER LINEAR EQUALITIES

The results in the previous section on congruences are now used to give polynomial algorithms for finding an integer solution to  $Ax = b$  or showing one does not exist. The motivation of our algorithm is based on Example 4.1 used in the introduction of this chapter.

**Example 4.16 (Example 4.1 continued)** *In this example illustrating Gaussian elimination the equality  $2x_1 + x_2 + x_3 = 2$  was used to project out variable  $x_1$  by making the substitution  $x_1 = (2 - x_2 - x_3)/2$  into the remaining equation(s). The remaining system is solved for  $x_2$  and  $x_3$ . Integer values for  $x_2$  and  $x_3$  do not guarantee an integer value for  $x_1$ . For this to be true  $(2 - x_2 - x_3)$  must be divisible by 2. That is, the additional condition  $x_2 + x_3 \equiv 2 \pmod{2}$  is required. This idea is used to replace a system of linear Diophantine equalities with a system of linear congruences.*

Without loss, assume  $A$  is an integer  $m \times n$  matrix of rank  $r$  and that  $b$  is an integer  $m$  vector. Also assume the columns of  $A$  are ordered so that the submatrix of consisting of columns  $1, \dots, r$  has rank  $r$ . Using row operations

and the methods described in Appendix B we can, in strongly polynomial time, bring the system  $Ax = b$  into the form

$$\begin{array}{llllll} \theta x_1 & +\hat{a}_{1,r+1}x_{r+1}+\cdots+\hat{a}_{1n}x_n & = & \hat{b}_1 \\ \theta x_2 & +\hat{a}_{2,r+1}x_{r+1}+\cdots+\hat{a}_{2n}x_n & = & \hat{b}_2 \\ \vdots & \vdots & \vdots & \vdots \\ \theta x_r & +\hat{a}_{r,r+1}x_{r+1}+\cdots+\hat{a}_{rn}x_n & = & \hat{b}_r \end{array} \quad (4.30)$$

where  $\theta$  is the determinant of submatrix of  $A$  defined by columns  $1, \dots, r$ , rows  $1, \dots, r$  and all  $\hat{a}_{ij}$  and  $\hat{b}_i$  are integers.

**Proposition 4.17** *Assume  $A$  is an integer  $m \times n$  matrix,  $b$  is an integer  $m$  vector and the submatrix of  $A$  consisting of columns  $1, \dots, m$  has rank  $m$ . There is an integer solution to  $Ax = b$  if and only if there is an integer solution to*

$$\sum_{j=m+1}^n \hat{a}_{ij}x_j \equiv \hat{b}_i \pmod{|\theta|}, \quad i = 1, \dots, m \quad (4.31)$$

where  $\theta$ ,  $\hat{a}_{ij}$ , and  $\hat{b}_i$  are defined in (4.30).

In the case of  $m = n$ , equation (4.31) reduces to the condition that the determinant  $\theta$  must divide  $\hat{b}_i$ ,  $i = 1, \dots, m$ . Thus, the problem of finding a solution (all solutions) to a system of linear equations reduces to finding a solution (all solutions) to a system of linear congruences.

**Example 4.18 (Example 4.1 continued.)** *The inverse of the matrix*

$$B = \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix} \quad \text{is} \quad B^{-1} = \begin{bmatrix} \frac{3}{2} & -\frac{1}{2} \\ -2 & 1 \end{bmatrix}$$

and the determinant of  $B$  is  $-2$ . Multiply the system of equations by  $-2B^{-1}$  and get

$$\begin{array}{lll} -2x_1 & -2x_3 & = 1 \\ -2x_2 & +2x_3 & = -6 \end{array}$$

This system has an integer solution if and only if there is a solution to the system

$$\begin{array}{ll} -2x_3 & \equiv 1 \pmod{2} \\ 2x_3 & \equiv -6 \pmod{2}. \end{array}$$

There is no feasible solution to the first congruence so there is no feasible solution to the Diophantine linear system.

Combining Proposition 4.15 and 4.17 gives the following result of Smith [408].

**Proposition 4.19 (Smith)** *If  $A$  is a rational  $m \times n$  matrix of rank  $r$  and  $b$  is a rational  $m$  vector, then there exist linearly independent vectors,  $x^1, \dots, x^q$  where  $q = n - r$  and a vector  $x^0$  such that  $x$  is an integer solution of  $Ax = b$  if and only if*

$$x = x^0 + G$$

where  $G$  is the lattice

$$G = \{x \mid x = z_1 x^1 + z_2 x^2 + \dots + z_q x^q, z_1, z_2, \dots, z_q \in \mathbb{Z}\}.$$

In developing algorithms for solving systems of linear congruences and linear integer equations our main concern has been ease of exposition and illustrating the power of projection and inverse projection. We have ignored finding the most efficient algorithms and implementation issues. For actual implementation and computational results see Bradley [70], Cabay and Lam [76], Chou and Collins [91] and Domich, Kannan, and Trotter [128].

## 4.5 INTEGER LINEAR INEQUALITIES: PROJECTION

The decision problem, “given rational  $A, b$ , is there an integer solution to  $Ax \geq b$ ,” is  $\mathcal{NP}$ -complete. See Appendix B. We first tackle this problem using the projection method. The ideas and results in this section are from Williams [450, 454] and [455]. Assume we are projecting out variable  $x_1$  and that  $a_{i1} > 0$  and  $a_{ik} < 0$ . Scaling rows  $i$  and  $k$  by  $a_{i1}$  and  $-a_{k1}$  respectively, gives

$$\begin{aligned} x_1 + (1/a_{i1}) \sum_{j=2}^n a_{ij} x_j &\geq b_i/a_{i1} \\ -x_1 - (1/a_{k1}) \sum_{j=2}^n a_{kj} x_j &\geq -b_k/a_{k1}. \end{aligned}$$

Then

$$(1/a_{i1})(b_i - \sum_{j=2}^n a_{ij} x_j) \leq x_1 \leq (1/a_{k1})(b_k - \sum_{j=2}^n a_{kj} x_j). \quad (4.32)$$

If  $x_1$  is an integer variable then (4.32) implies

$$x_1 \geq \lceil (1/a_{i1})(b_i - \sum_{j=2}^n a_{ij}x_j) \rceil. \quad (4.33)$$

This is equivalent to the condition that there is an integer variable  $s_1 \in \{0, 1, \dots, a_{i1} - 1\}$  such that

$$x_1 \geq (1/a_{i1})(b_i - \sum_{j=2}^n a_{ij}x_j) + s_1/a_{i1} \quad (4.34)$$

$$\sum_{j=2}^n a_{ij}x_j \equiv b_i + s_1 \pmod{a_{i1}}. \quad (4.35)$$

Further, in order to satisfy both inequalities in (4.32) it is necessary that

$$(b_i - \sum_{j=2}^n a_{ij}x_j) + s_1 \leq (a_{i1}/a_{k1})(b_k - \sum_{j=2}^n a_{kj}x_j). \quad (4.36)$$

**Example 4.20** This example is from Williams [455].

$$\begin{aligned} 7x_1 - 2x_2 &\geq 0 \\ -3x_1 + x_2 &\geq 0 \\ x_1, x_2 &\in \mathbb{Z} \end{aligned}$$

Eliminate  $x_1$ . Divide the first equation by 7 and the second by 3.

$$x_1 - \frac{2}{7}x_2 \geq 0, \quad -x_1 + \frac{1}{3}x_2 \geq 0$$

This implies

$$\frac{2}{7}x_2 \leq x_1 \leq \frac{1}{3}x_2.$$

Then the projected system is

$$\begin{aligned} \frac{2}{7}x_2 + \frac{1}{7}s_1 &\leq \frac{1}{3}x_2 \\ 2x_2 + s_1 &\equiv 0 \pmod{7} \\ x_2 \in \mathbb{Z}, s_1 \in \{0, 1, \dots, 6\} \end{aligned}$$

Clearly the integer case is not as “nice” as the continuous case. In the continuous case projecting a polyhedron into a lower dimensional space is again a polyhedron in the lower dimensional space. However, projecting  $P \cap \mathbb{Z}^n$ , where  $P$  is a rational polyhedron in  $\mathbb{R}^n$  into a lower dimensional space does not result in the intersection of a polyhedron with an integer lattice. Example 4.20 is such an example. The set in this example is a polyhedral monoid, but the projection into the  $x_2$  space gives the points  $\{0, 3, 6, 7, 9, 10, \dots\}$  which cannot be expressed as the intersection of  $\mathbb{Z}$  with a polyhedron in  $x_2$  and is therefore not a polyhedral monoid. Another difference is that in the continuous case, projection gives the extreme rays and extreme points of a “dual” polyhedron. Inverse projection applied to the dual problem gives the extreme points and rays of the primal problem. This duality does not extend to the integer case. However, projection does provide results on optimal value functions similar to Propositions 2.47 and 2.48. First, a definition is required.

Blair and Jeroslow [61] define the class  $\mathcal{C}_m$  of *m-dimensional Chvátal functions* as the smallest class  $\mathcal{C}$  of functions with the properties:

1.  $f \in \mathcal{C}$  if  $f(v) = z^\top v$ ,  $z$  rational and  $v = (v_1, v_2, \dots, v_m)$ ;
2.  $f, g \in \mathcal{C}$ ,  $\alpha, \beta \geq 0$ ,  $\alpha, \beta$  rational implies  $\alpha f + \beta g \in \mathcal{C}$ ;
3.  $f \in \mathcal{C}$  implies  $[f] \in \mathcal{C}$  where  $[f](v) = [f(v)]$ .

The class of  $\mathcal{G}_m$  *m-dimensional Gomory functions* is the smallest class  $\mathcal{C}$  with properties 1-3 plus the property

$$4 \quad f, g \in \mathcal{C} \text{ implies } \max\{f, g\} \in \mathcal{C}.$$

Blair and Jeroslow show that for the integer program

$$\min\{c^\top x \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}, \quad (4.37)$$

there is a Gomory function  $F(b)$  such that (4.37) is feasible if and only if  $F(b) \leq 0$  (note the analogy with Proposition 2.47) and that if this problem is feasible there is a Gomory function  $G(b)$  such that

$$G(b) = \min\{c^\top x \mid Ax = b, x \geq 0, x \in \mathbb{Z}^n\}.$$

Williams [454] has shown that this value function is also derived by repeated application of the variable elimination method just demonstrated. Blair and

Jeroslow also show the converse result that for a given Gomory function  $G$ , there is an integer program which is feasible for all integer vectors  $b$  and has  $G(b)$  as the optimal value function. These are very powerful and deep results. Blair [60] has more recently extended these results to the mixed case. Earlier Jeroslow [246] and Wolsey [461, 460] showed that if (4.37) has an optimal solution, then the optimal value function is an optimal solution to the dual problem

$$\begin{aligned} \max \quad & f(b) \\ \text{s.t.} \quad & f(a^j) \leq c_j, \quad j = 1, \dots, n \end{aligned}$$

where  $f$  is a subadditive function on the monoid generated by the columns of  $A$ , and in particular,  $f$  is a Chvátal function. Unfortunately, unlike the continuous case we cannot take the “dual of the dual” and get the primal problem back. In the continuous case, we can replace dual constraints with primal variables. There is not a way to replace subadditive functions with variables and the duality breaks down.

## 4.6 INTEGER LINEAR INEQUALITIES: INVERSE PROJECTION

The linear space  $L = \{x \in \mathbb{R}^n \mid Ax = 0\}$  is generated by  $x^1, \dots, x^q$  linearly independent vectors where  $q = n - \text{rank}(A)$ . Similarly, the affine space  $H = \{x \in \mathbb{R}^n \mid Ax = b\}$  can be represented as

$$H = \{x \in \mathbb{R}^n \mid Ax = b\} = x^0 + L.$$

Earlier in this chapter we extended these results and showed that when  $A$  is a rational matrix the additive group

$$G = \{x \in \mathbb{R}^n \mid Ax = 0\} \cap \mathbb{Z}^n$$

is finitely generated, i.e. a lattice. We also proved a result due to Smith that when  $A$  and  $b$  are rational,

$$\begin{aligned} \{x \in \mathbb{R}^n \mid Ax = b\} \cap \mathbb{Z}^n &= x^0 + G \\ &= x^0 + \sum_{i=1}^q z_i x^i \end{aligned}$$

where  $z_1, \dots, z_q \in \mathbb{Z}$  and  $x^0, x^1, \dots, x^q \in \mathbb{Z}^n$ . We now extend these results for cones and polyhedrons. In Chapter 3, we proved the Minkowski finite basis theorem for the polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ . That is,

$$P = \text{conv}\{x^1, \dots, x^q\} + \text{cone}\{x^{q+1}, \dots, x^r\}.$$

In this section we develop integer analogs for the finite basis theorems for cones and polyhedrons. First consider the integer monoid

$$M = \{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\} \quad (4.38)$$

where  $A$  is an  $m \times n$  rational matrix. If the integrality condition  $x \in \mathbb{Z}^n$  is replaced with  $x \in \mathbb{R}^n$  then  $M$  is a polyhedral cone. An integer polyhedral monoid is the intersection of a polyhedral cone with  $\mathbb{Z}^n$ . We first show that  $M$  is finitely generated. By finitely generated, we mean there are finite number of elements of  $M$  called a basis, such that every element in  $M$  can be written as an nonnegative integer combination of basis elements. This is done by applying the inverse projection method developed in Chapter 3 to the system  $Ax = 0, x \geq 0$ . Begin with row one. Assume the columns are ordered so that  $a_{1i} > 0$  for  $i = 1, \dots, n_1$ ,  $a_{1,n_1+j} < 0$ , for  $j = 1, \dots, n_2$  and  $a_{1j} = 0$ , for  $j = n_1 + n_2 + 1, \dots, n$ . Since  $A$  is a rational matrix, without loss, assume all  $a_{ij}$  are integers. The first constraint  $\sum_{j=1}^n a_{1j}x_j = 0$  in  $M$  is eliminated using the variable transformation

$$x_i = \sum_{j=1}^{n_2} y_{ij}/a_{1i}, \quad i = 1, \dots, n_1 \quad (4.39)$$

$$x_{n_1+j} = -\sum_{i=1}^{n_1} y_{ij}/a_{i,n_1+j}, \quad j = 1, \dots, n_2. \quad (4.40)$$

Now substitute out the  $x$  variables in the system  $Ax = 0$  using (4.39)-(4.40). The system in the new variables is

$$\begin{aligned} & \sum_{i=1}^{n_1} \left( a_{ki} \sum_{j=1}^{n_2} y_{ij}/a_{1i} \right) - \sum_{j=1}^{n_2} \left( a_{k,n_1+j} \sum_{i=1}^{n_1} y_{ij}/a_{i,n_1+j} \right) \\ & + \sum_{l=n_1+n_2+1}^n a_{kl}x_l = 0, \quad k = 1, \dots, m. \end{aligned} \quad (4.41)$$

We guarantee nonnegativity of  $x$  by requiring  $y$  to be nonnegative.

$$y_{ij} \geq 0, \quad i = 1, \dots, n_1, \quad j = 1, \dots, n_2 \quad x_i \geq 0, \quad i = n_1 + n_2 + 1, \dots, n. \quad (4.42)$$

We have shown the following result (repeated here for completeness) in Chapter 3.

**Proposition 4.21** *If  $C = \{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$  then  $\bar{x} \in C$  if and only if there is a  $\bar{y}$  such that  $(\bar{x}, \bar{y})$  is feasible to (4.39)-(4.42).*

In words, Proposition 4.21 states the  $C$  is the result of projecting out the  $y$  variables from (4.39)-(4.42). But what about the condition that  $x$  is an integer vector? Since the  $a_{1j}$  are not necessarily  $\pm 1$  the integrality of  $y_{ij}$  does not imply integrality of  $x_i$  in (4.39)-(4.40). However, if the following congruence conditions on  $y_{ij}$

$$\sum_{j=1}^{n_2} y_{ij} \equiv 0 \pmod{a_{1i}}, \quad i = 1, \dots, n_1 \quad (4.43)$$

$$\sum_{i=1}^{n_1} y_{ij} \equiv 0 \pmod{a_{1,n_1+j}}, \quad j = 1, \dots, n_2. \quad (4.44)$$

are satisfied, then  $y_{ij}$  integer implies  $x_i$  integer.

**Proposition 4.22** *If  $M = \{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\}$  and  $A$  is a rational matrix, then  $\bar{x}$  is in the monoid  $M$  if and only if there is a  $\bar{y}$  such that  $(\bar{x}, \bar{y})$  is feasible to (4.39)-(4.44).*

**Proof:** If  $A$  is a rational matrix, we can assume without loss that  $A$  is an integer matrix. If  $(\bar{x}, \bar{y})$  satisfies (4.39)-(4.42) then from Proposition 4.21,  $\bar{x} \in C$ . But if  $\bar{y}$  satisfies (4.43)-(4.44) then  $\bar{x}$  is integer and therefore in the monoid  $M$ .

Next assume  $\bar{x} \in M$ . The crucial idea is to show that there is an integer solution  $(\bar{x}, \bar{y})$  to (4.39)-(4.40) for  $x$  fixed at  $\bar{x}$ . At this point, we refer the reader to Subsection 1.3.5 in Chapter 1 where the transportation linear program, (*TLP*) is introduced. See also, Figure 3.2 in Chapter 3. Set up a transportation linear program with supply nodes  $i = 1, \dots, n_1$  with a supply of  $a_{1i}\bar{x}_i$  and demand nodes  $j = 1, \dots, n_2$  with demand  $a_{1,n_1+j}\bar{x}_j$ . Since  $A$  is an integer matrix and  $\bar{x}$  is integer there is an integer solution  $\bar{y}_{ij}$  to this transportation linear program. The rational for this is given in Chapter 14. Furthermore, the integrality of  $\bar{x}$  and  $\bar{y}$  and (4.39)-(4.40) imply (4.41)-(4.44). Thus,  $(\bar{x}, \bar{y})$  is feasible to (4.39)-(4.44).  $\square$

**Proposition 4.23** *If  $A$  is a rational matrix, the integer monoid  $M = \{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\}$  is finitely generated.*

**Proof:** Repeated application of inverse projection will eliminate all constraints and replace  $M$  with a system of congruences

$$\sum_{i \in I_k} d_{ki} u_i \equiv 0 \pmod{\theta_k} \quad k = 1, \dots, t \quad (4.45)$$

and nonnegativity constraints

$$u_i \geq 0 \text{ and integer for all } i. \quad (4.46)$$

By repeated application of Proposition 4.22 the monoid defined by (4.45)-(4.46) is identical to the monoid  $M$ . Let  $\theta$  be the least common multiple of  $\theta_1, \dots, \theta_t$ . Any nonnegative solution to (4.45) is also a solution after reduction mod  $\theta$ . Therefore, all of the solutions to (4.45)-(4.46) with  $u_i \leq \theta$  generate the monoid. This is a finite number.  $\square$

**Example 4.24 (Example 4.20 continued.)** In the standard form for inverse projection the problem is

$$\begin{array}{cccccc} 7x_1 & -2x_2 & -x_3 & & = & 0 \\ -3x_1 & +x_2 & -x_4 & & = & 0 \\ x_1, & x_2, & x_3, & x_4 & \in & \mathbb{Z}_+ \end{array}$$

Project out the first constraint by using the variable substitution

$$x_1 = \frac{1}{7}y_1 + \frac{1}{7}y_2, \quad x_2 = \frac{1}{2}y_1, \quad x_3 = y_2.$$

The new system is (after making the coefficients integer)

$$\begin{array}{cccccc} y_1 & -6y_2 & -14x_4 & = & 0 \\ y_1, & y_2, & x_4 & \in & \mathbb{Z}_+. \end{array}$$

In order to guarantee the integrality of the  $x$  variables, we add the congruences

$$\begin{aligned} y_1 + y_2 &\equiv 0 \pmod{7} \\ y_1 &\equiv 0 \pmod{2}. \end{aligned}$$

Now eliminate the second constraint with the variable substitution

$$y_1 = v_1 + v_2, \quad y_2 = \frac{1}{6}v_1, \quad x_4 = \frac{1}{14}v_2.$$

In order to guarantee integrality of the  $y$  variables, we add the congruences

$$\begin{aligned} v_1 &\equiv 0 \pmod{6} \\ v_2 &\equiv 0 \pmod{14}. \end{aligned}$$

It is also necessary to express the previous congruences in terms of the new variables. The previous congruences become

$$\begin{aligned} 7v_1 + 6v_2 &\equiv 0 \pmod{42} \\ v_1 + v_2 &\equiv 0 \pmod{2}. \end{aligned}$$

We have expressed the congruences so that all coefficients are integers. All the nonnegative solutions to these four congruence systems are nonnegative integer combinations of the vectors  $(6, 0)$  and  $(0, 14)$ . The  $v$  variables are transformed back into  $x$  by

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{7} & \frac{1}{7} \\ \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ \frac{1}{6} & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Under this transformation the generators  $(6, 0)$  and  $(0, 14)$  in  $v$ -space become  $(1, 3)$  and  $(2, 7)$  in  $x$ -space.

In Proposition 4.23 we work with a cone in the standard form  $\{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$ . The result obviously extends to cones not in standard form.

**Corollary 4.25** *If  $A$  is a rational matrix, the integer monoid  $M = \{x \in \mathbb{Z}^n \mid Ax \geq 0, x \geq 0\}$  is finitely generated.*

Proposition 4.23 is an old result that dates back to Hilbert [229]. In fact, the finite set for an integer monoid which generates the entire monoid is often called a *Hilbert basis*. See also Jeroslow [245] for results and background on finite basis results for integer monoids. Given that there exists a finite basis (which is minimal relative to taking subsets) for an integer polyhedral monoid, is that basis unique? A sufficient condition to guarantee uniqueness of a minimal Hilbert basis is that the cone be pointed. The following corollary is left as an exercise.

**Corollary 4.26** *If  $A$  is a rational matrix and the cone  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$  is pointed, then the integer monoid  $M = C \cap \mathbb{Z}^n$  has a unique minimal basis generating  $M$ .*

Just as the finite basis result for cones is extended to polyhedrons, we extend the finite basis results from monoids to include the set  $P \cap \mathbb{Z}^n$  where  $P$  is a rational polyhedron. In particular, we show  $P \cap \mathbb{Z} = M + T$  where  $T$  is a finite set of integer vectors and  $M$  is an integer polyhedral monoid.

**Theorem 4.27 (Integer Finite Basis Theorem)** *If  $A$  is a rational matrix,  $b$  a rational vector,  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$  and  $\Gamma = P \cap \mathbb{Z}^n$ , then*

1.  $\Gamma = T + M$  where  $T = \{x^1, \dots, x^q\} \subset \mathbb{Z}^n$  is a finite set of integer vectors and  $M$  is the integer monoid  $\{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\}$ ;
2.  $\text{conv}(\Gamma) = \text{conv}\{x^1, \dots, x^q\} + \text{cone}\{x^{q+1}, \dots, x^r\}$  where  $x^{q+1}, \dots, x^r \in \mathbb{Z}^n$  and generate  $M$ .

**Proof:** First prove part 1. Consider the integer polyhedral monoid

$$\hat{M} = \{(x, x_0) \in \mathbb{Z}^{n+1} \mid Ax - bx_0 = 0, x, x_0 \geq 0\}.$$

By Proposition 4.23, the integer polyhedral monoid  $\hat{M}$  is finitely generated. Let  $i = 1, \dots, q$  index the generators  $x^i$  of the monoid where  $x_0^i = 1$  and  $i = q+1, \dots, r$  index the generators  $x^i$  where  $x_0^i = 0$ . Let  $\bar{x} \in \Gamma$ . Then  $(\bar{x}, 1) \in \hat{M}$ . Thus  $(\bar{x}, 1)$  is a nonnegative integer combination of elements from the finite set which generates  $\hat{M}$ . Clearly, none of vectors used to generate  $(\bar{x}, 1)$  can have an  $x_0$  component other than 0 or 1 and there is exactly one vector from the generator set with an  $x_0$  component of 1. Thus, there are nonnegative integers,  $\bar{z}_i, i = 1, \dots, r$  such that

$$\bar{x} = \sum_{i=1}^q \bar{z}_i x^i + \sum_{i=q+1}^r \bar{z}_i x^i, \quad \sum_{i=1}^q \bar{z}_i = 1. \quad (4.47)$$

Conversely, any vector generated by (4.47) is in  $\Gamma$ . Let  $T$  be the set generators of  $\hat{M}$  for which  $x_0 = 1$  and  $M$  the integer monoid in  $\mathbb{Z}^n$  generated by the generators of  $\hat{M}$  for which  $x_0 = 0$  and part 1 is proved.

Now prove part 2 which says that  $\text{conv}(\Gamma)$  is a rational polyhedron. First prove  $\text{conv}(\Gamma) \subseteq \text{conv}(T) + \text{cone}(M)$ . Let  $\bar{x} \in \text{conv}(\Gamma)$ . Show  $\bar{x} \in \text{conv}(T) + \text{cone}(M)$ . By the Carathéodory (see Appendix A) theorem there are a finite number of points  $s^1, \dots, s^t$  in  $\Gamma$  such that  $\bar{x} = \sum_{i=1}^t \lambda_i s^i$ ,  $\sum_{i=1}^t \lambda_i = 1$  and  $\lambda_i \geq 0$ ,  $i = 1, \dots, t$ . From part 1,  $s^i \in \Gamma$  implies  $s^i = x^{k(i)} + \sum_{l=q+1}^r z_{il} x^l$  where  $x^{k(i)} \in T$ ,  $x^l \in M$  and the  $z_{il}$  are nonnegative integers. Then,

$$\begin{aligned} \bar{x} &= \sum_{i=1}^t \lambda_i s^i \\ &= \sum_{i=1}^t \lambda_i \left( x^{k(i)} + \sum_{l=q+1}^r z_{il} x^l \right) \\ &= \sum_{i=1}^t \lambda_i x^{k(i)} + \sum_{l=q+1}^r \left( \sum_{i=1}^t \lambda_i z_{il} \right) x^l. \end{aligned}$$

Since  $(\sum_{i=1}^t \lambda_i x_{il}) \geq 0$  for all  $l$  and  $\sum_{i=1}^t \lambda_i = 1$ ,  $\bar{x} \in \text{conv}(T) + \text{cone}(M)$ .

Next, show  $\text{conv}(T) + \text{cone}(M) \subseteq \text{conv}(\Gamma)$ . The result follows from observing that  $\text{cone}(M) = \text{conv}(M)$  and  $\text{conv}(T) + \text{conv}(M) \subseteq \text{conv}(\Gamma)$ .  $\square$

Theorem 4.27 is extremely significant. If  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$  and  $\Gamma = P \cap \mathbb{Z}^n$  then

$$\min\{c^\top x \mid x \in \Gamma\} = \min\{c^\top x \mid x \in \text{conv}(\Gamma)\}.$$

By part 2 of Theorem 4.27,  $\text{conv}(\Gamma)$  is finitely generated and is therefore a polyhedron. Because the generators are integer vectors the polyhedron  $\text{conv}(\Gamma)$  is an integer polyhedron. A rational polyhedron is an *integer polyhedron* if and only if every face contains an integer vector. (In the case of a pointed polyhedron, it is an integer polyhedron if and only if every extreme point is an integer vector.) Therefore, an *integer* linear program can, in theory, always be solved as a linear program once we have the inequality characterization of the underlying integer polyhedron. Indeed, in later chapters we try to characterize, at least partially, the inequalities that define  $\text{conv}(P)$ . This Theorem is also used later in a number of important ways in formulating integer programming models.

Part 2 of Theorem 4.27 is strengthened by observing that for rational matrices  $A$ ,

$$\text{cone}(\{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\}) = \{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}. \quad (4.48)$$

Then the  $x^i$ ,  $i = q+1, \dots, r$  used in part 2 can be restricted to the extreme rays of the cone  $\{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$ . If  $A$  is an  $m \times n$  matrix of rank  $m$ , the maximum number of extreme rays possible in the cone  $\{x \mid Ax = 0, x \geq 0\}$  is  $\binom{n}{m}$ . This number is independent of the data in the  $A$  matrix. Unfortunately, there is not a similar result for the number of generators of an integer polyhedral monoid. Consider the following example.

**Example 4.28** Let<sup>2</sup>

$$M = \{(x_1, x_2) \in \mathbb{Z}^2 \mid ax_1 - x_2 \geq 0, x_1, x_2 \geq 0\} \quad (4.49)$$

where  $a$  is a positive integer. Convert this to the standard form for inverse projection and work with the system

$$\begin{array}{ccc|c} ax_1 & -x_2 & -x_3 & = & 0 \\ x_1, & x_2, & x_3 & \geq & 0 \end{array}$$

---

<sup>2</sup>This example was conveyed to me by W.R. Pulleyblank who got it from A. Schrijver.

*Make the variable substitution*

$$x_1 = (y_1 + y_2)/a, \quad x_2 = y_1, \quad x_3 = y_2$$

*into the constraint  $ax_1 - x_2 - x_3 = 0$ . We are left with monoid*

$$\begin{aligned} y_1 + y_2 &\equiv 0 \pmod{a} \\ y_1, y_2 &\geq 0. \end{aligned}$$

*Considering only solutions with  $y_1, y_2 \leq a$  gives the following basis for the monoid*

$$(y_1, y_2) = (a, 0), (a-1, 1), (a-2, 2), \dots, (1, a-1), (0, a). \quad (4.50)$$

*The corresponding solutions back in the original  $(x_1, x_2)$  space are*

$$(x_1, x_2) = (1, a), (1, a-1), (1, a-2), \dots, (1, 1), (1, 0) \quad (4.51)$$

*The extreme rays of the polyhedral cone given by the conic hull of the integer polyhedral monoid (4.49) are  $(1, 0)$  and  $(1, a)$ . However, none of the points  $(1, 1), \dots, (1, a-1)$  are integer combinations of these extreme rays. For the monoid illustrated in (4.49),  $a+1$  elements are required in the basis. There is no bound on the number of elements by the rows or the number of variables defining the monoid.*

*This example also shows that it is not valid to replace  $M$  in part 1 of the integer finite basis theorem with the extreme rays of the cone  $\{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$ .*

Many important applications are actually formulated as mixed integer programming models so an analog to the integer finite basis theorem is needed when a proper subset of the variables are continuous.

**Lemma 4.29** *If  $A, B$  are rational matrices,  $b$  a rational vector and*

$$\Gamma = \{(x, y) \mid Ax + By = b, x \geq 0, y \in \mathbb{Z}_+^l\}, \quad (4.52)$$

*then the projection into the  $y$  variable space*

$$\text{proj}_{(y)}(\Gamma) = \{y \in \mathbb{Z}_+^l \mid \text{there is an } x \text{ such that } (x, y) \in \Gamma\}$$

*is the intersection of  $\mathbb{Z}_+^l$  with a rational polyhedron in  $\mathbb{R}^l$ .*

**Proof:** Project out the continuous variables in  $\Gamma$ . This gives a set of rational inequalities in  $\mathfrak{R}_+^l$ . The intersection of this polyhedron with  $\mathcal{Z}^l$  gives  $\text{proj}_{(y)}(\Gamma)$ .  $\square$

**Theorem 4.30 (Mixed Integer Finite Basis Theorem)** *If*

$$\Gamma = \{(x, y) \mid Ax + By = b, x \geq 0, y \in \mathcal{Z}_+^l\}, \quad (4.53)$$

where  $A, B$  are rational matrices and  $b$  is a rational vector, then  $\text{conv}(\Gamma)$  is a rational polyhedron.

**Proof:** Show  $\text{conv}(\Gamma)$  is finitely generated. Construct the generators as follows. By Lemma 4.29,  $\text{proj}_{(y)}(\Gamma)$  is the intersection of a rational polyhedron with  $\mathcal{Z}_+^l$  so by Theorem 4.27,  $\text{proj}_{(y)}(\Gamma) = T + M$  where  $T$  is a finite set of integer vectors  $y^1, \dots, y^q$  and  $M$  is an integer monoid with generators  $y^{q+1}, \dots, y^r$ .

For each integer vector,  $y^i$ ,  $i = 1, \dots, q$ , in  $T$  define

$$\begin{aligned} \Gamma(y^i) &:= \{(x, y) \mid Ax + By = b, y = y^i + \sum_{k=q+1}^r z_k y^k, x \geq 0, \\ &\quad z_k \in \mathcal{Z}_+, k = q+1, \dots, r\}. \end{aligned} \quad (4.54)$$

Let  $\bar{\Gamma}(y^i)$  denote the set obtained from  $\Gamma(y^i)$  by replacing the integrality condition  $z_k \in \mathcal{Z}_+$  with  $z_k \geq 0$ . Each  $\bar{\Gamma}(y^i)$  is a polyhedron since applying projection to the  $z_k$  variables results in a finite system of linear inequalities. This implies each  $\bar{\Gamma}(y^i)$  is finitely generated. The polyhedrons  $\bar{\Gamma}(y^i)$  differ only in the right hand sides so they all have the same generators for their recession cones. Let  $C$  be the finite set of these generators. Without loss,  $(\bar{x}, \bar{y}) \in C$  implies  $\bar{y}$  is an integer vector.

Let  $T_i$  be the finite set of extreme points of  $\bar{\Gamma}(y^i)$ . If  $(\hat{x}, \hat{y}) \in \Gamma$  then there is an  $i$  such that  $(\hat{x}, \hat{y}) \in \Gamma(y^i) \subseteq \bar{\Gamma}(y^i)$ . This implies

$$(\hat{x}, \hat{y}) \in \text{conv}(T_i) + \text{cone}(C).$$

This implies  $\text{conv}(\Gamma) \subseteq \text{conv}(\cup_{i=1}^q T_i) + \text{cone}(C)$ .

If  $(\bar{x}, \bar{y})$  is an extreme point of  $\bar{\Gamma}(y^i)$ , then  $\bar{y}$  is an integer vector. See Exercise 4.22. This implies that any point from  $T_i$  plus a point from  $C$  is in  $\Gamma$ . Therefore,  $\text{conv}(\Gamma)$  is a convex combination of points from  $\cup_{i=1}^q T_i$  plus a conic combination of points in  $\text{cone}(C)$ .  $\square$

We have not shown that there is a finite basis for generating every element of  $\Gamma$ . However,  $\text{conv}(\Gamma)$  is a polyhedron and therefore finitely generated so we still call the theorem the *mixed integer finite basis theorem*.

The projection of the finitely generated set

$$\{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i, \sum_{i=1}^q z_i = 1, z_i \geq 0 \text{ } i = 1, \dots, r\}$$

into  $x$ -space is a polyhedron. In the case where this is a cone we have the result of Weyl. Conversely, using inverse projection we have shown the result of Minkowski that any polyhedron (or polyhedral cone) is finitely generated. Projection and inverse projection are dual processes. Unfortunately, this Weyl-Minkowski duality does not exist in the integer case. By Proposition 4.23 the polyhedral integer monoid  $\{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\}$  is finitely generated. However, the finitely generated monoid,  $M = \{x \mid x = Az, z \in \mathbb{Z}^n\}$ , where  $A$  is an  $m \times n$  integer matrix is not necessarily an integer polyhedral monoid. Consider the following trivial example.

### Example 4.31

$$M = \{x \mid x = 2z, z \in \mathbb{Z}\}$$

*Clearly there  $M$  cannot be polyhedral monoid since there is no inequality system in  $\mathbb{R}^1$  containing the even positive integers but not the odd positive integers.*

Kirby and Williams [265] have shown that any finitely generated integer monoid is the projection of the intersection of a polyhedron with the integer lattice. However, Blair and Jeroslow [61] have shown the following Weyl like theorem.

**Theorem 4.32 (Blair and Jeroslow)** *If  $M = \{x \mid x = Az, z \in \mathbb{Z}^n\}$  where  $A$  is an  $m \times n$  integer matrix, then there exist  $m$ -dimensional Chvátal functions  $f_1, \dots, f_p$  such*

$$M = \{y \in \mathbb{Z}^m \mid f_i(y) \geq 0, i = 1, \dots, p\}.$$

## 4.7 CONCLUSION

Projecting out the variables of a primal linear program generates all the extreme points and rays of the dual polyhedron and recession cone. See Proposition 2.31.

Applying inverse projection to a linear program gives all of the extreme points and extreme rays of the primal polyhedron and recession cone. Unfortunately, this Weyl-Minkowski duality does not exist for integer programming. Inverse projection is much “nicer” than projection for integer programming. We have shown how to get an integer equivalent of the Minkowski theorem using inverse projection, but a result similar to Weyl is not possible for finitely generated integer monoids. Without this Weyl-Minkowski like duality for integer linear programming a duality theory is pretty much doomed and the elegant results such as those of Blair, Jeroslow and Wolsey are probably the best one can hope for.

## 4.8 EXERCISES

- 4.1 Prove that the Algorithm 4.3 applied to positive integers  $a_1, a_2$  has complexity  $O(\max\{\log_2(a_1), \log_2(a_2)\})$ .
- 4.2 Prove Lemma 4.2.
- 4.3 Find an integer solution to the equation  $8216x_1 + 1508x_2 = 260$  using the Euclidean algorithm.
- 4.4 Prove Proposition 4.7.
- 4.5 Using the methods developed in this chapter, find an integer solution to the system

$$\begin{aligned} 343x_1 + 17x_2 + 16x_3 + 45x_4 &= 55 \\ 12x_1 + 10x_2 + 5x_3 + 75x_4 &= -16 \end{aligned}$$

or show that one does not exist.

- 4.6 Give an algorithm for calculating the gcd of two integers  $a_1, a_2$  which *does not* use a divide instruction. Your algorithm must rely solely on
  - a. testing whether a number is even or odd;
  - b. shifting a binary representation of an even number to the right;
  - c. subtraction.

Here is another way to think about this problem: can you write a program in C or FORTRAN for finding  $\gcd(a_1, a_2)$  which never requires a division. See Stein [411].

4.7 Prove that given two congruences

$$x \equiv f \pmod{m_1}, \quad x \equiv g \pmod{m_2}, \quad (4.55)$$

there exist multipliers  $\lambda_1, \lambda_2$  such that (4.55) is equivalent to

$$x \equiv \lambda_1 f + \lambda_2 g \pmod{\text{lcm}(m_1, m_2)}, \quad 0 \equiv f - g \pmod{\gcd(m_1, m_2)}.$$

See Williams [454]. This result is known as the Chinese remainder theorem.

4.8 If  $A$  is an  $m \times n$  rational matrix,  $M = \{x \in \mathbb{Z}^n \mid Ax = 0, x \geq 0\}$  and  $C = \{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$ , then

$$\text{conv}(M) = \text{cone}(M) = C.$$

4.9 Prove that if  $T$  is a nonsingular unimodular matrix then  $T^{-1}$  is an integer matrix.

4.10 Find the Hermite normal form of the matrix

$$A = \begin{bmatrix} 343 & 17 & 16 & 45 \\ 12 & 10 & 5 & 75 \end{bmatrix}.$$

4.11 Take the matrix

$$A = \begin{bmatrix} 343 & 17 & 16 & 45 \\ 12 & 10 & 5 & 75 \end{bmatrix}.$$

and find an integer solution to  $Ax = b$  where  $b_1 = 55$  and  $b_2 = -16$  or show one does not exist.

4.12 Prove that the set of integer solutions to the system  $Ax = b$  with  $A, b$  rational is characterized by

$$\{x = \bar{x} + z_1 x^1 + \cdots + z_t x^t \mid z_1, \dots, z_t \in \mathbb{Z}\}$$

where the  $x^i$  are linearly independent and  $t = n - \text{rank}(A)$ .

4.13 Prove that the Hermite normal form of a matrix is unique.

4.14 In the proof of the finite basis theorem for integer polyhedral monoids, where is the hypothesis that  $A$  is a rational matrix used?

4.15 Prove that if  $P = \{x \in \mathbb{Z}^n \mid Ax = b, x \geq 0\}$ , then

$$\min\{c^\top x \mid x \in P\} = \min\{c^\top x \mid x \in \text{conv}(P)\}.$$

- 4.16 Show that there are integers  $s_1, \dots, s_5$  which give a solution of  $x = (0, -30, 90, 15, 0)$  for the system (4.20). Show that there are integers  $s_1, \dots, s_5$  which give a solution of  $x = (-2, -31, 2, 0, 0)$  for the system (4.29).
- 4.17 Show that solution matrix in (4.20) can be obtained from the solution matrix in (4.29) by unimodular column operations.
- 4.18 Prove Proposition 4.15.
- 4.19 Prove Proposition 4.17.
- 4.20 Prove Proposition 4.19.
- 4.21 Prove that if  $\bar{\Gamma} = \{(x, y) \mid Ax + By = b, x, y \geq 0\}$ , and  $\hat{y}$  is an extreme ray of the recession cone of the projected polyhedron  $\text{proj}_{(y)}(\bar{\Gamma})$ , then there is an  $\hat{x}$  such that  $(\hat{x}, \hat{y})$  is an extreme ray of the recession cone of the polyhedron defined by  $\bar{\Gamma}$ .
- 4.22 Let  $\bar{\Gamma}(y^i)$  denote the set obtained from  $\Gamma(y^i)$  by replacing the integrality condition  $z^k \in \mathcal{Z}_+$  with  $z_k \geq 0$  where  $\Gamma(y^i)$  is defined in (4.54). Show that if  $(\bar{x}, \bar{y})$  is an extreme point of  $\bar{\Gamma}(y^i)$ , then  $\bar{y}$  is an integer vector.
- 4.23 Prove Corollary 4.25.
- 4.24 Prove Corollary 4.26.

## **PART III**

---

### **ALGORITHMS**

---

# THE SIMPLEX ALGORITHM

## 5.1 INTRODUCTION

In Chapter 2 a projection algorithm was given for solving linear programs. Unfortunately, this projection algorithm is not practical because the number of constraints resulting from projecting out variables is generally an exponential function (and possibly doubly exponential) of the number of variables projected out. In this chapter we develop the simplex algorithm for solving linear programs. Although not a polynomial algorithm, the simplex algorithm due to Dantzig [108] is very effective and was the first practical algorithm for solving linear programs. The motivation behind the simplex algorithm is given in Section 5.2. The simplex algorithm is based on pivoting and this idea is introduced in Section 5.3. Important enhancements on the simplex algorithm are given in Sections 5.4 and 5.5. These enhancements have contributed dramatically to the success of simplex in solving real world models. In Section 5.6 rules are given to guarantee finite convergence of the simplex algorithm. The complexity of the simplex method is addressed in Section 5.7. Concluding remarks are given in Section 5.8. Exercises are provided in Section 5.9.

## 5.2 MOTIVATION

A linear program in *standard form* is

$$(LP) \quad \begin{aligned} & \min c^T x \\ \text{s.t. } & Ax = b \\ & x \geq 0 \end{aligned}$$

where  $A$  is an  $m \times n$  matrix with rank  $m \leq n$ ,  $b$  is an  $m$  vector,  $c$  is an  $n$  vector and  $x$  is an  $n$  vector of variables. Any linear program is easily put into standard form by either adding or subtracting nonnegative *slack variables* to convert an inequality constraint into an equality constraint. If a variable  $x_j$  is unrestricted, it is converted into two nonnegative variables by the transformation  $x_j = x'_j - x''_j$ .

In order to motivate the simplex method, write linear program ( $LP$ ) as

$$z_0 - c^\top x = 0 \quad (5.1)$$

$$Ax = b \quad (5.2)$$

$$x \geq 0 \quad (5.3)$$

This format is identical to that used for solving a linear program using projection except that the objective function row is written as  $z_0 - c^\top x = 0$  instead of  $z_0 - c^\top x \geq 0$ . Since the problem is a minimization,  $z_0 - c^\top x \geq 0$  is always tight, and there is no loss of generality in restricting this row to be an equality constraint. By assumption,  $A$  has rank  $m$ . Then  $m$  variables can be projected out by applying Gaussian elimination to the  $Ax = b$  constraints.

Before applying Gaussian elimination, we briefly digress and introduce notation used in this chapter. If  $A$  has rank  $m$ , there is an ordered index set  $B$ , of cardinality  $m$ , such that the columns of  $A$  indexed by  $B$  are linearly independent. The columns indexed by  $B$  are a *basis* and the variables  $x_i$ ,  $i \in B$  are called *basic variables*. By  $x_B$  we mean an  $m$  component vector of variables with components  $x_{B,i}$  for  $i = 1, \dots, m$ . We often say variable  $x_{B,i}$  is basic in row  $i$ . Let  $A_B$  be the  $m \times m$  submatrix of  $A$  consisting of the columns of  $A$  indexed by  $B$ . The variables not indexed  $B$  are called *nonbasic variables*. The nonbasic variables are indexed by  $N$ . The matrix  $A_N$  is the  $m \times (n-m)$  submatrix of  $A$  consisting of the columns of  $A$  indexed by  $N$ . Similarly,  $x_N$  is an  $(n-m)$  component vector of nonbasic variables.

Given a basis  $B$ , the constraints  $Ax = b$  can be rewritten as  $A_B x_B + A_N x_N = b$ . Multiplying through by  $A_B^{-1}$  gives

$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b.$$

Given basis  $B$ , a *basic solution* is

$$\begin{bmatrix} \bar{x}_B \\ \bar{x}_N \end{bmatrix} = \begin{bmatrix} A_B^{-1} b \\ 0 \end{bmatrix}.$$

Thus,  $\bar{x}_B$  represents a specific setting of the basic variables  $x_B$ . Similarly for  $\bar{x}_N$  and  $x_N$ . The solution  $\bar{x}_B = A_B^{-1} b$ ,  $\bar{x}_N = 0$  is a *basic feasible solution* if  $\bar{x}_B$  is nonnegative.

Now reconsider system (5.1)-(5.3). Given a basis  $B$ , this system is equivalent to

$$z_0 - [c_B^\top \ c_N^\top] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = 0 \quad (5.4)$$

$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b \quad (5.5)$$

$$x_B, x_N \geq 0. \quad (5.6)$$

Substitute  $x_B = A_B^{-1}b - A_B^{-1}A_N x_N$  from (5.5) into (5.4) and (5.6) and simplify. This projection into the space of the nonbasic variables gives the equivalent linear program

$$z_0 - \bar{w}_N^\top x_N = c_B^\top A_B^{-1} b \quad (5.7)$$

$$A_B^{-1} A_N x_N \leq A_B^{-1} b \quad (5.8)$$

$$x_N \geq 0. \quad (5.9)$$

where

$$\bar{w}_N^\top := c_N^\top - c_B^\top A_B^{-1} A_N. \quad (5.10)$$

In linear programming jargon, the  $\bar{w}_N$  defined in (5.10) are called *reduced costs*. See Section 3.4 in Chapter 3 for a discussion of the reduced cost. The projected linear program in the space of nonbasic variables reveals important information about the basic solution associated with the basis  $B$ . Given a basic feasible solution  $\bar{x}_B = A_B^{-1}b$ ,  $\bar{x}_N = 0$  with solution value  $c_B^\top A_B^{-1}b$ , can this solution value be improved? If  $\bar{w}_q < 0$ , for a  $q \in N$ , then (5.7) implies  $z_0$  must decrease by  $\bar{w}_q$  for each unit  $x_q$  is increased above its current 0 value. Clearly, it is desirable to increase  $x_q$  as much as possible. How much can  $x_q$  be increased without making one of the  $A_B^{-1} A_N x_N \leq A_B^{-1} b = \bar{x}_B$  constraints in (5.8) infeasible? Denote by  $\bar{a}_{iq}$  the element in row  $i$ , column  $q$  of  $A_B^{-1} A_N$ . Column  $q$  of the matrix  $A_B^{-1} A_N$  is  $\bar{a}^q$ . Define

$$i^* := \operatorname{argmin} \{\bar{x}_{B_i} / \bar{a}_{iq} \mid \bar{a}_{iq} > 0, i = 1, \dots, m\}. \quad (5.11)$$

$$\alpha_P := \bar{x}_{B_{i^*}} / \bar{a}_{i^* q} \quad (5.12)$$

where the  $\operatorname{argmin}$  of a function is the element of the function domain which minimizes the function. The quantity  $\alpha_P$  represents the maximum possible increase in  $x_q$  without violating the inequalities  $A_B^{-1} A_N x_N \leq A_B^{-1} b$ . If the variable  $x_q$  is increased by  $\alpha_P$  then the objective function value of  $c_B^\top A_B^{-1} b$  is decreased to  $c_B^\top A_B^{-1} b + \alpha_P \bar{w}_q$ . The test used to determine  $i^*$  in equation (5.11) is called the *minimum ratio test*. If  $i^*$  is undefined, that is, there are no strictly positive  $\bar{a}_{iq}$ , then the linear program is unbounded. The significance of the upper case  $P$  used in the subscript of  $\alpha$  is explained later.

**Proposition 5.1** Given a basis  $B$ , if the basic solution  $(\bar{x}_B, \bar{x}_N)$  is feasible and if  $\bar{w}_N \geq 0$ , then this solution is an optimal solution to  $(LP)$ . If there is a  $q \in N$  such that  $\bar{w}_q < 0$  and  $\bar{a}^q \leq 0$ , then the linear program  $(LP)$  is unbounded.

**Proof:** The original linear program (5.4)-(5.6) is equivalent to the projected linear program (5.7)-(5.9) in the sense that  $\bar{x}_N = 0$  is an optimal solution to (5.7)-(5.9) if and only if  $(\bar{x}_B, \bar{x}_N)$  is an optimal solution to (5.4)-(5.6). Next, observe that the nonnegativity of  $\bar{x}_B$  implies that  $\bar{x}_N = 0$  is a feasible solution to the projected linear program. Since the linear program is a minimization problem,  $\bar{w}_N \geq 0$  implies  $\bar{x}_N = 0$  is an optimal solution to the projected linear program and the first part of the proposition is proved. The second part of the proposition follows from the fact that  $\bar{w}_q < 0$  and  $\bar{a}^q \leq 0$  implies that the projected linear program is unbounded.  $\square$

**Example 5.2** The example linear program is

$$\begin{array}{lllll} \min & 2x_1 & -3x_2 & & \\ & & x_2 & \geq & 1 \\ & x_1 & +x_2 & \geq & 2 \\ & -.5x_1 & +x_2 & \leq & 8 \\ & -x_1 & +x_2 & \leq & 6 \end{array}$$

In standard form this linear program is

$$\begin{array}{lllll} \min & 2x_1 & -3x_2 & & \\ & & x_2 & -x_3 & = 1 \\ & x_1 & +x_2 & -x_4 & = 2 \\ & -.5x_1 & +x_2 & +x_5 & = 8 \\ & -x_1 & +x_2 & +x_6 & = 6 \end{array}$$

Define  $B = \{1, 2, 5, 6\}$  and  $N = \{3, 4\}$ . The system written as  $A_B x_B + A_N x_N = b$  is

$$\begin{array}{lllll} & x_2 & -x_3 & = & 1 \\ x_1 & +x_2 & -x_4 & = & 2 \\ -.5x_1 & +x_2 & +x_5 & = & 8 \\ -x_1 & +x_2 & +x_6 & = & 6 \end{array}$$

The basis matrix  $A_B$  and its inverse  $A_B^{-1}$  are

$$A_B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -.5 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix} \quad A_B^{-1} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1.5 & .5 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix}.$$

Multiplying the constraints  $A_B x_B + A_N x_N = b$  by  $A_B^{-1}$  gives

$$\begin{array}{rccccc} x_1 & +x_3 & -x_4 & = & 1 \\ x_2 & -x_3 & & = & 1 \\ x_5 & +1.5x_3 & -.5x_4 & = & 7.5 \\ x_6 & +2x_3 & -x_4 & = & 6 \end{array}$$

Projecting the linear program onto  $(x_3, x_4)$  space by substituting out  $x_B = A_B^{-1}b - A_B^{-1}A_N x_N$  gives

$$\begin{array}{lcl} z_0 + 5x_3 - 2x_4 & = & -1 \\ x_3 - x_4 & \leq & 1 \\ -x_3 & \leq & 1 \\ 1.5x_3 - .5x_4 & \leq & 7.5 \\ 2x_3 - x_4 & \leq & 6 \\ x_3, x_4 & \geq & 0 \end{array}$$

For every unit  $x_3$  is increased, the objective function value  $z_0$  decreases by 5 units. It is desirable to make  $x_3$  as large as possible. The minimum ratio test gives

$$\alpha_P = \min\{(1/1), (7.5/1.5), (6/2)\} = 1.$$

Increasing  $x_3$  from 0 to 1 decreases the objective function value from  $-1$  to  $-6$ . Increasing  $x_3$  from 0 to 1 makes the constraint  $x_3 - x_4 \leq 1$  tight. This constraint corresponds to the nonnegativity requirement on basic variable  $x_1$ . Therefore increasing the nonbasic variable  $x_3$  from 0 to 1 drives the value of basic variable  $x_1$  from 1 to 0. The variables indexed by  $\{3, 2, 5, 6\}$  are a new basis.

### 5.3 PIVOTING

Now, consider what is happening in terms of all the variables, not just the space of projected variables. The system including both basic and nonbasic variables is

$$\begin{array}{lcl} z_0 - \bar{w}_N^\top x_N & = & c_B^\top \bar{x}_B \\ x_B + A_B^{-1} A_N x_N & = & \bar{x}_B \\ x_B, x_N & \geq & 0. \end{array}$$

If  $q \in N$  and  $x_q$  is increased from 0 the relation  $x_B + A_B^{-1}A_Nx_N = \bar{x}_B$  implies component  $x_{B,i}$  of  $x_B$  decreases by  $\bar{a}_{B,iq}$  for each unit  $x_q$  is increased. If  $x_q$  is increased by  $\alpha_P$  where  $\alpha_P$  is defined in (5.12) then, the new solution  $\hat{x}_B, \hat{x}_N$  is

$$\begin{bmatrix} \hat{x}_B \\ \hat{x}_N \end{bmatrix} = \begin{bmatrix} \bar{x}_B \\ \bar{x}_N \end{bmatrix} + \alpha_P \begin{bmatrix} \Delta x_B \\ \Delta x_N \end{bmatrix}$$

where

$$\Delta x_B = \begin{bmatrix} -\bar{a}_{1q} \\ -\bar{a}_{2q} \\ \vdots \\ -\bar{a}_{mq} \end{bmatrix}$$

and  $\Delta x_N$  is an  $(n - m)$  component unit vector with the 1 in the component corresponding to variable  $x_q$ .

If  $\alpha_P > 0$ , then taking a step of length  $\alpha_P$  in direction  $(\Delta x_B, \Delta x_N)^\top$  drives basic variable  $x_{B,*}$  to zero and nonbasic variable  $q$  is now positive. There is a new basis  $\hat{B} = B \setminus \{B_{i,*}\} \cup \{q\}$  and a new set of nonbasic variables  $\hat{N} = N \setminus \{q\} \cup \{B_{i,*}\}$ . By construction, variable  $x_q$  has a positive coefficient in row  $i^*$ , the row in which variable  $x_{B,*}$  is basic. This implies that  $A_{\hat{B}}$  has rank  $m$  and that  $\hat{B}$  is indeed a basis.

Replacing one variable in the basis with another variable is called *pivoting*. In Example 5.2 variable  $x_3$  pivots into the basis and variable  $x_1$  pivots out of the basis. If variable  $x_q$  is selected to enter the basis, and  $x_{B,*}$  is selected to leave the basis, then before this change occurs row  $i^*$  is

$$x_{B_{i^*}} + \bar{a}_{i^*q}x_q + \sum_{\substack{j \in N \\ j \neq q}} \bar{a}_{i^*j}x_j = \bar{x}_{B_{i^*}}.$$

The coefficient  $\bar{a}_{i^*q}$  is called the *pivot element*. If  $q$  is in the new basis  $\hat{B}$  then it is necessary to project out variable  $x_q$  using row  $i^*$ . Solving for  $x_q$  gives

$$x_q = (\bar{x}_{B_{i^*}} / \bar{a}_{i^*q}) - x_{B_{i^*}} / \bar{a}_{i^*q} - \sum_{\substack{j \in N \\ j \neq q}} (\bar{a}_{i^*j} / \bar{a}_{i^*q})x_j. \quad (5.13)$$

Substituting for  $x_q$  in the remaining equations

$$x_{B_i} + \sum_{j \in N} \bar{a}_{ij}x_j = \bar{x}_{B_i}, \quad i = 1, \dots, m, \quad i \neq i^*$$

and objective function row

$$z_0 - \sum_{j \in N} \bar{w}_j x_j = c_B^\top \bar{x}_B$$

gives

$$\begin{aligned} z_0 - (\bar{w}_q / \bar{a}_{i^* q}) x_{B_{i^*}} &= \sum_{\substack{j \in N \\ j \neq q}} (\bar{w}_j - \bar{w}_q (\bar{a}_{i^* j} / \bar{a}_{i^* q})) x_j \\ &= c_B^\top \bar{x}_B + \bar{w}_q (\bar{x}_{B_{i^*}} / \bar{a}_{i^* q}) \end{aligned} \quad (5.14)$$

$$\begin{aligned} x_{B_i} - (\bar{a}_{iq} / \bar{a}_{i^* q}) x_{B_{i^*}} &+ \sum_{j \in N} (\bar{a}_{ij} - \bar{a}_{iq} (\bar{a}_{i^* j} / \bar{a}_{i^* q})) x_j \\ &= \bar{x}_{B_i} - (\bar{a}_{iq} / \bar{a}_{i^* q}) \bar{x}_{B_{i^*}}, \quad i = 1, \dots, m, \quad i \neq i^* \end{aligned} \quad (5.15)$$

$$x_{B_{i^*}} / \bar{a}_{i^* q} + x_q + \sum_{\substack{j \in N \\ j \neq q}} (\bar{a}_{i^* j} / \bar{a}_{i^* q}) x_j = \bar{x}_{B_{i^*}} / \bar{a}_{i^* q} \quad (5.16)$$

Thus, the coefficients of  $A_{\hat{B}}^{-1} A_{\hat{N}}$  are given by

$$\hat{a}_{ij} = \bar{a}_{ij} - \bar{a}_{iq} (\bar{a}_{i^* j} / \bar{a}_{i^* q}), \quad i = 1, \dots, m, \quad i \neq i^*, \quad j \in \hat{N}, \quad j \neq B_{i^*} \quad (5.17)$$

$$\hat{a}_{i^* j} = (\bar{a}_{i^* j} / \bar{a}_{i^* q}), \quad j \in \hat{N}, \quad j \neq B_{i^*} \quad (5.18)$$

$$\hat{a}_{i^* B_{i^*}} = -(\bar{a}_{iq} / \bar{a}_{i^* q}), \quad i = 1, \dots, m, \quad i \neq i^* \quad (5.19)$$

$$\hat{a}_{i^* B_{i^*}} = (1 / \bar{a}_{i^* q}). \quad (5.20)$$

This system (5.14)-(5.16) is called a *simplex tableau*. Substituting out  $x_q$  is equivalent to performing row operations on the simplex tableau. The substitution method or Gaussian elimination is equivalent to pivoting on  $\bar{a}_{i^* q}$ . The *pivot operation* consists of dividing the *pivot row* (5.13) by  $\bar{a}_{i^* q}$  and performing the necessary row operations to make all elements of column  $q$  a 0 except in the pivot row. This pivot operation is equivalent to substitution and also gives (5.17)-(5.20).

**Example 5.3 (Example 5.2 continued.)** For the basis  $B = \{1, 2, 5, 6\}$ , the objective function row and system  $x_B + A_B^{-1} A_N x_N = A_B^{-1} b$  is

$$\begin{array}{rccccccccc} z_0 & & +5x_3 & -2x_4 & = & -1 \\ x_1 & & +x_3 & -x_4 & = & 1 \\ x_2 & & -x_3 & & = & 1 \\ x_5 & & +1.5x_3 & -0.5x_4 & = & 7.5 \\ x_6 & & +2x_3 & -x_4 & = & 6 \end{array}$$

Pivot on the element corresponding variable  $x_3$  in row 1 in the system  $x_B + A_B^{-1}A_Nx_N = A_B^{-1}b$ . The new system is

$$\begin{array}{rccccc} z_0 & -5x_1 & & +3x_4 & = & -6 \\ & x_1 & & +x_3 & -x_4 & = & 1 \\ & x_1 & x_2 & & -x_4 & = & 2 \\ -1.5x_1 & & x_5 & & +x_4 & = & 6 \\ -2x_1 & & x_6 & & +x_4 & = & 4 \end{array}$$

The new basis is  $\hat{B} = \{3, 2, 5, 6\}$ . Exchange columns 1 and 5. The new system  $x_{\hat{B}} + A_{\hat{B}}^{-1}A_Nx_{\hat{N}} = A_{\hat{B}}^{-1}b$  is

$$\begin{array}{rccccc} z_0 & & -5x_1 & +3x_4 & = & -6 \\ x_3 & & +x_1 & -x_4 & = & 1 \\ x_2 & & +x_1 & -x_4 & = & 2 \\ x_5 & & -1.5x_1 & +x_4 & = & 6 \\ x_6 & & -2x_1 & +x_4 & = & 4 \end{array}$$

The *simplex algorithm* is the pivoting process repeated until an optimal solution is found.

#### Algorithm 5.4 (Simplex Algorithm)

**Step 1: (Initialization)** Initialize with a basis matrix  $A_B$  such that  $A_B^{-1}b \geq 0$  and create the initial tableau

$$\begin{aligned} z_0 - \bar{w}_N x_N &= c_B^\top \bar{x}_B \\ x_B + A_B^{-1}A_N x_N &= \bar{x}_B \end{aligned}$$

**Step 2: (Pivot Column Selection)** If  $\bar{w}_N \geq 0$ , stop, the current solution is optimal; else, select  $q \in N$  such that  $\bar{w}_q < 0$ .

**Step 3: (Minimum Ratio Test and Pivot Row Selection)** Find the maximum allowable increase of variable  $x_q$ . If  $\bar{a}_{iq}^q \leq 0$ , stop the linear program is unbounded. Otherwise, perform the minimum ratio test

$$\begin{aligned} i^* &\leftarrow \operatorname{argmin} \{\bar{x}_B_i / \bar{a}_{iq} \mid \bar{a}_{iq} > 0, i = 1, \dots, m\} \\ \alpha_P &\leftarrow \bar{x}_{B_{i^*}} / \bar{a}_{i^*q} \end{aligned}$$

**Step 4: (Pivot and Update Tableau)** Pivot on element  $\bar{a}_{i^*q}$  and update the tableau using (5.17)-(5.20).

**Step 5: (Update the Basis and Reorder the Tableau)** *Update the basis*

$$\begin{aligned} B &\leftarrow (B \setminus \{B_i\}) \cup \{q\} \\ N &\leftarrow (N \setminus \{q\}) \cup \{B_i\}. \end{aligned}$$

and reorder the tableau so the basic variables are the first  $m$  variables in the tableau. Go to **Step 2**.

Calculating the reduced costs is often called *pricing*. The columns are *priced out* to give the reduced costs. Two important issues in terms of efficiency are i) which variable with a negative reduced cost should be selected in Step 3, and ii) if there are ties in the minimum ratio test how should the ties be broken. Dantzig [108] suggests selecting the entering variable by picking the variable with the most negative reduced cost, i.e.

$$q = \operatorname{argmin} \{\bar{w}_j \mid \bar{w}_j < 0, j \in N\}.$$

This is called the *Dantzig rule*. The problem of selecting an entering variable is discussed in greater depth in the next chapter. We also postpone a more detailed discussion of the ratio test until the next chapter.

**Example 5.5 (Example 5.2 continued)**

### Iteration 2

**Step 1: (Initialization)** At the start of the second iterations the basis is  $B = \{3, 2, 5, 6\}$  and the tableau for this basis is given below.

$$\begin{array}{rccccccccc} z_0 & & -5x_1 & +3x_4 & = & -6 \\ x_3 & & +x_1 & -x_4 & = & 1 \\ x_2 & & +x_1 & -x_4 & = & 2 \\ x_5 & & -1.5x_1 & +x_4 & = & 6 \\ x_6 & & -2x_1 & +x_4 & = & 4 \end{array}$$

**Step 2: (Pivot Column Selection)** Variable  $x_4$  is the only nonbasic variable with a negative reduced cost.

**Step 3: (Minimum Ratio Test)** Perform the minimum ratio test to see how much  $x_4$  can be increased.

$$\min \{(6/1) = 6, (4/1) = 4\} = 4$$

Variable  $x_6$  leaves the basis.

**Steps 3 and 4: (Pivot and Update the Basis)** Pivot  $x_4$  into the basis and pivot  $x_6$  out of the basis.

$$\begin{array}{rccccccccc} z_0 & & x_1 & -3x_6 & = & -18 \\ x_3 & & -x_1 & +x_6 & = & 5 \\ x_2 & & -x_1 & +x_6 & = & 6 \\ x_5 & & +.5x_1 & -x_6 & = & 2 \\ x_4 & & -2x_1 & +x_6 & = & 4 \end{array}$$

$$B \leftarrow \{3, 2, 5, 4\}$$

### Iteration 3

**Step 2: (Pivot Column Selection)** Variable  $x_1$  is the only nonbasic variable with a negative reduced cost and should again enter the basis.

**Step 3: (Minimum Ratio Test)** Perform the minimum ratio test to see how much  $x_1$  can be increased.

$$\min \{2/.5\} = 4$$

Variable  $x_5$  leaves the basis. It is the only variable with a positive coefficient in the column corresponding to  $x_1$ .

**Steps 3 and 4: (Pivot and Update the Basis)**

$$\begin{array}{rccccccccc} z_0 & & -2x_5 & -x_6 & = & -22 \\ x_3 & & +2x_5 & -x_6 & = & 9 \\ x_2 & & +2x_5 & -x_6 & = & 10 \\ x_1 & & +2x_5 & -2x_6 & = & 4 \\ x_4 & & +4x_5 & -3x_6 & = & 12 \end{array}$$

$$B \leftarrow \{3, 2, 1, 4\}$$

### Iteration 4

**Step 2: (Pivot Column Selection)** All of the reduced costs of the nonbasic variables are positive. Bringing either  $x_5$  or  $x_6$  into solution will increase the objective function value. The current solution is optimal by Proposition 5.1.

It is important to prove that the simplex algorithm is finite. Clearly, there are a finite number of bases. If  $A$  is an  $m \times n$  matrix of rank  $m$ , then an upper bound on the number of basic feasible solutions is  $\binom{n}{m}$ . Therefore, the only way the simplex algorithm can fail to be finite is for it to repeat basic feasible solutions. We give a sufficient condition to prevent this.

Recall from Chapter 2 that a solution  $\bar{x}$  of the linear program  $\min \{c^\top x \mid Ax \geq b\}$  where  $A$  is an  $m \times n$  matrix is *primal degenerate* (or *degenerate*) if the rank of the submatrix of  $A$  defined by the tight constraints, i.e.  $(a^i)^\top \bar{x} = b_i$  has rank strictly less than the number of tight constraints. Consider a linear program in standard form  $\min \{c^\top x \mid Ax = b, x \geq 0\}$  where  $A$  is an  $m \times n$  matrix with rank  $m$ . If the definition of degeneracy is applied to a linear program in standard form, then a primal solution  $\bar{x}$  is degenerate if and only if the number of strictly positive variables is less than  $m$ . To see that this corresponds to the degeneracy concept defined previously, observe that for any solution  $\bar{x}$ , the tight constraints are  $A\bar{x} = b$  and  $\bar{x}_i = 0$  for all the zero components of  $\bar{x}$ . If  $k$  is the number of positive components in  $\bar{x}$  and  $k < m$ , then the number of tight constraints is  $m + (n - k) > n$ . This must be a degenerate solution in  $\mathbb{R}^n$  since the rank of the tight constraints cannot exceed  $n$ . A linear program in standard form is *nondegenerate* if all of the basic feasible solutions are nondegenerate. Nondegeneracy is a sufficient condition for convergence of the simplex algorithm.

**Proposition 5.6** *If the linear program  $\min \{c^\top x \mid Ax = b, x \geq 0\}$  is nondegenerate and has an optimal solution, then starting with a basic feasible solution, the simplex algorithm will converge to the optimal solution in a finite number of pivots.*

**Proof:** By hypothesis the polyhedron  $P = \{x \mid Ax = b, x \geq 0\}$  is not empty and therefore has a finite number of extreme points. Then Steps 1 - 4 are executed a finite number of times if no extreme point solution is repeated. By the nondegeneracy assumption, variable  $\bar{x}_j$  basic implies that  $\bar{x}_j > 0$ . Assume nonbasic variable  $q$  is selected in Step 2. By hypothesis, the linear program has an optimal solution and therefore is not unbounded and there is a finite  $\alpha_P > 0$  implied by the minimum ratio test. The new objective function value after pivoting in variable  $x_q$  is  $z_0 + \alpha_P \bar{w}_q < z_0$ . Since the objective function value strictly decreases at each pivot there is no repetition of extreme points. Therefore, simplex converges in a finite number of pivots. This solution must be optimal because the simplex algorithm applied to a linear program with an optimal solution terminates only if all of the reduced costs are nonnegative. In this case, by Proposition 5.1 the current basic solution is optimal.  $\square$

Convergence of the simplex algorithm for degenerate linear programs is treated in Section 5.6 of this chapter.

We conclude this section by providing a geometric interpretation of basic feasible solutions. The basic feasible solution property is clearly an algebraic one. Namely, given an index set  $B$ , of cardinality  $m$ ,  $A_B$  has rank  $m$  and  $A_B^{-1}b$  is nonnegative. However, if  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ , a basic feasible solution  $\bar{x}$  corresponds to an extreme point of  $P$ . If  $\bar{x} \in P$ , then  $\bar{x}$  is an *extreme point* of  $P$  if and only if there do not exist distinct  $x^1, x^2 \in P$  and  $\lambda \in (0, 1)$  such that  $\bar{x} = \lambda x^1 + (1 - \lambda)x^2$ . See Appendix A. The equivalence of extreme points and basic feasible solution is given in the following proposition which is left as an exercise.

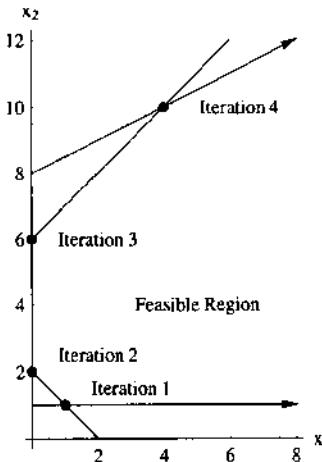
**Proposition 5.7 (Basic Feasible Solution)** *The vector  $\bar{x} \in \mathbb{R}^n$  is a basic feasible solution of the system  $Ax = b, x \geq 0$  if and only if it is an extreme point of  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ .*

If a linear program has an optimal solution, and the simplex algorithm does not cycle, then the simplex algorithm will terminate with an optimal basic feasible solution. Then Propositions 5.6 and 5.7 imply that if there is an optimal solution to a nondegenerate linear program there is an optimal extreme point solution. Later we show how to remove the degeneracy assumption and maintain convergence of the simplex algorithm. This gives an algorithmic proof to the fundamental theorem of linear programming given in Proposition 3.4. The extreme point solution corresponding at the start of each simplex iteration in Example 5.2 is illustrated in Figure 5.1.

## 5.4 REVISED SIMPLEX

The simplex algorithm as described in the previous section is not practical from a computational standpoint because it does not take advantage of sparsity of the constraint matrix  $A$  that is almost always present in any real application. In Example 5.2 after the first pivot the nonbasic columns are 100% dense. Even if the  $A$  matrix is very sparse, the matrix  $A_B^{-1}A_N$  is typically very dense. If there are thousands of nonbasic variables, as there are in real applications, this makes it impossible to work with a dense matrix  $A_B^{-1}A_N$  since the number of nonzero elements is  $O(nm)$ . In addition to the storage problems, if  $A_B^{-1}A_N$  is dense the number of calculations required in updating  $A_B^{-1}A_N$  in Step 4 of

Figure 5.1 Extreme Points for Example 5.2



the simplex algorithm is also  $O(nm)$ . Working with these nonzero elements is not only time consuming, but causes considerable roundoff error. The revised simplex algorithm maintains the sparsity of the nonbasic columns. In Step 4 of simplex algorithm, the dense matrix  $A_B^{-1}A_N$  is calculated by pivoting on the basic variables. This dense matrix is required in two steps of the simplex algorithm. First, the reduced costs  $\bar{w}_N^T = c_N^T - c_B^T A_B^{-1} A_N$  are used to select the entering variable in Step 2. Then in Step 3, the exit variable is determined using the ratio test. The ratio test requires  $\bar{a}^q$  where  $x_q$  is the entering variable. The selection of the entering and exit variable however really only requires knowing  $A_B^{-1}$ . First consider the reduced cost calculation. Assume we are at the start of a generic simplex iteration and the new basis is  $\hat{B} = B \setminus \{B_{i^*}\} \cup \{q\}$  where  $B$  was the basis at the start of the previous iteration. Then from (5.14), for  $j \in \hat{N}$  and  $j \neq B_{i^*}$  the reduced cost  $\hat{w}_j$  of variable  $j$ , corresponding to basis  $\hat{B}$  is

$$\hat{w}_j = \bar{w}_j - \bar{w}_q \left( \frac{\bar{a}_{i^* j}}{\bar{a}_{i^* q}} \right)$$

where  $\bar{w}_j$  is the reduced cost of variable  $x_j$  corresponding to basis  $B$ . The entire matrix  $A_B^{-1}A_N$  is not required. Only elements  $\bar{a}_{i^* j}$  and  $\bar{a}_{i^* q}$  in row  $i^*$  are required. But row  $i^*$  of  $A_B^{-1}A_N$  can be calculated when needed from row  $i^*$  of  $A_B^{-1}$ . Similarly, only  $\bar{a}^q$  is needed for the ratio test, not the entire matrix  $A_B^{-1}A_N$ . Column  $\bar{a}^q$  is  $A_B^{-1}a^q$ . Therefore, at each iteration of the simplex algorithm it is necessary to only know  $A_B^{-1}$ , not the entire tableau  $A_B^{-1}A_N$ .

Given  $A_B$ , calculating  $A_B^{-1}$  requires  $O(m^3)$  arithmetic operations. It is not necessary to recalculate  $A_B^{-1}$  from scratch each time. If  $\hat{B} = B \setminus \{B_{i^*}\} \cup \{q\}$  and variable  $x_{B,i^*}$  is basic in row  $i^*$ , then

$$A_{\hat{B}} = A_B + (a^q - A_B e^{i^*})(e^{i^*})^\top.$$

The inverse of  $A_{\hat{B}}$  is given by the Sherman-Morrison-Woodbury *rank one update* formula (see Householder [237]).

**Lemma 5.8** *If*

$$A_{\hat{B}} = A_B + (a^q - A_B e^{i^*})(e^{i^*})^\top, \quad (5.21)$$

$A_{\hat{B}}^{-1}$  is nonsingular, and  $\bar{a}_{i^*,q} \neq 0$ , then

$$\begin{aligned} A_{\hat{B}}^{-1} &= A_B^{-1} + \frac{A_B^{-1}(a^q - A_B e^{i^*})(e^{i^*})^\top A_B^{-1}}{(e^{i^*})^\top A_B^{-1} a^q} \\ &= \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*,q}} \right) A_B^{-1}. \end{aligned} \quad (5.22)$$

**Proof:** If  $\bar{a}^q = A_B^{-1} a^q$ , the expression (5.22) becomes  $A_{\hat{B}}^{-1} = EA_B^{-1}$  where

$$E = \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*,q}} \right). \quad (5.23)$$

Since  $A_B$  and  $A_{\hat{B}}$  differ only in column  $i^*$ . Then  $A_B^{-1} A_{\hat{B}}$  and the identity matrix differ only in column  $i^*$  and

$$EA_B^{-1} A_{\hat{B}} = E \begin{bmatrix} e^1 & e^2 & \cdots & e^{i^*-1} & \bar{a}^q & e^{i^*+1} & \cdots & e^m \end{bmatrix}. \quad (5.24)$$

Since  $(e^{i^*})^\top e^i = 0$  for  $i \neq i^*$ , and  $(e^{i^*})^\top \bar{a}^q = \bar{a}_{i^*,q}$ , using the definition of  $E$  in (5.23), equation (5.24) becomes

$$\begin{aligned} &E \begin{bmatrix} e^1 & e^2 & \cdots & e^{i^*-1} & \bar{a}^q & e^{i^*+1} & \cdots & e^m \end{bmatrix} \\ &= \begin{bmatrix} e^1 & e^2 & \cdots & e^{i^*-1} & \bar{a}^q & e^{i^*+1} & \cdots & e^m \end{bmatrix} \\ &\quad - \begin{bmatrix} 0 & 0 & \cdots & 0 & (\bar{a}^q - e^{i^*}) & \cdots & 0 \end{bmatrix} = I, \end{aligned}$$

therefore,  $EA_B^{-1} = A_{\hat{B}}^{-1}$ .  $\square$

A more general version of the Sherman-Morrison-Woodbury formula is left as an exercise. The  $E$  matrix differs from the identity only in column  $i^*$  so the

update  $A_{\hat{B}}^{-1} = EA_B^{-1}$  requires work at most  $O(m^2)$ . If column  $i^*$  has  $d < m$  nonzero elements the work reduces to  $O(md)$ . Although, as we show in the next section, we do not explicitly store  $A_B^{-1}$  as an  $m \times m$  matrix, this update method is crucial to the revised simplex algorithm.

The rank one update has an important duality interpretation. For an arbitrary basis  $B$ , write the standard form linear program ( $LP$ ) as

$$(LP) \quad \begin{aligned} \min \quad & c_B^\top x_B + c_N^\top x_N \\ \text{s.t.} \quad & A_B x_B + A_N x_N = b \\ & x_B, x_N \geq 0. \end{aligned}$$

Let  $u$  be the dual variables on the constraint  $A_B x_B + A_N x_N = b$ ,  $w_B$  the dual variables on the  $x_B \geq 0$  constraints and  $w_N$  the dual variables on the  $x_N \geq 0$  constraints. The dual linear program is

$$(DLP) \quad \begin{aligned} \max \quad & b^\top u \\ \text{s.t.} \quad & A_B^\top u + w_B = c_B \\ & A_N^\top u + w_N = c_N \\ & w_B, w_N \geq 0. \end{aligned}$$

If  $\bar{u}^\top = c_B^\top A_B^{-1}$ ,  $\bar{w}_B = 0$  and  $\bar{w}_N = c_N - \bar{u}^\top A_N$ , then this is a dual feasible solution if  $\bar{w}_N$  is nonnegative. (We write the dual solution as a row vector  $\bar{u}^\top = c_B^\top A_B^{-1}$ , rather than  $\bar{u} = (A_B^{-1})^\top c_B$ , as the form  $c_B^\top A_B^{-1}$  is more suggestive of how we actually calculate the dual variables. See Section 5.5.) This is the termination criterion for the simplex algorithm. Thus, at each step the simplex algorithm is calculating a potential dual solution in addition to a feasible primal solution. Also,  $\bar{w}_B = 0$  and  $\bar{x}_N = 0$  so the primal-dual solution calculated by simplex satisfies complementary slackness at each iteration. Therefore, the simplex algorithm termination criterion that  $\bar{w}_N \geq 0$  is equivalent to dual feasibility. When the primal simplex algorithm terminates with an optimal primal solution, an optimal primal *and* dual solution is at hand since a primal-dual feasible solution satisfying complementary slackness is generated. See Corollary 2.34 in Chapter 2.

The rank one method of updating  $A_{\hat{B}}^{-1}$  can be used to update the dual solution at each simplex iteration. First update  $\hat{u}^\top = c_{\hat{B}}^\top A_{\hat{B}}^{-1}$  from  $\bar{u}$  and  $A_{\hat{B}}^{-1}$ . The new cost vector  $c_{\hat{B}}$  is related to the previous cost vector  $c_B$  by  $c_{\hat{B}}^\top = c_B^\top + (c_q - c_{i^*})(e^{i^*})^\top$ . Define  $(z^{i^*})^\top$  to be row  $i^*$  of  $A_B^{-1}$ . That is,

$$(z^{i^*})^\top := (e^{i^*})^\top A_B^{-1}. \quad (5.25)$$

Then

$$\begin{aligned}
 \hat{u}^\top &= c_B^\top A_B^{-1} = c_B^\top E A_E^{-1} \\
 &= \left( c_B^\top + (c_q - c_{i^*})(e^{i^*})^\top \right) \left( A_E^{-1} - \frac{(\bar{a}^q - e^{i^*})(z^{i^*})^\top}{\bar{a}_{i^*q}} \right) \\
 &= \bar{u}^\top - \frac{(\bar{u}^\top a^q - c_{i^*})(z^{i^*})^\top}{\bar{a}_{i^*q}} + (c_q - c_{i^*})(z^{i^*})^\top \\
 &\quad - \frac{(c_q - c_{i^*})(e^{i^*})^\top (\bar{a}^q - e^{i^*})(z^{i^*})^\top}{\bar{a}_{i^*q}} \\
 &= \bar{u}^\top - \frac{(\bar{u}^\top a^q - c_{i^*})(z^{i^*})^\top}{\bar{a}_{i^*q}} + \frac{(c_q - c_{i^*})(z^{i^*})^\top}{\bar{a}_{i^*q}} \\
 &= \bar{u}^\top + \frac{(c_q - \bar{u}^\top a^q)(z^{i^*})^\top}{\bar{a}_{i^*q}} = \bar{u}^\top + \frac{\bar{w}_q}{\bar{a}_{i^*q}} (z^{i^*})^\top.
 \end{aligned}$$

Using  $\hat{u}^\top = \bar{u}^\top + \frac{\bar{w}_q}{\bar{a}_{i^*q}} (z^{i^*})^\top$ , it follows that

$$\hat{w}_N^\top = c_N^\top - \hat{u}^\top A_N = \bar{w}_N^\top - \frac{\bar{w}_q}{\bar{a}_{i^*q}} (z^{i^*})^\top A_N. \quad (5.26)$$

Since  $(z^{i^*})^\top$  is row  $i^*$  of  $A_B^{-1}$ ,  $(z^{i^*})^\top A_N$  is row  $i^*$  of the updated tableau  $A_B^{-1} A_N$  and the reduced cost update formula (5.26) based on the rank one update is identical to the reduced cost update formula (5.14) based on pivoting. Define

$$\alpha_D := \bar{w}_q / \bar{a}_{i^*q}. \quad (5.27)$$

Then the new dual solution  $(\hat{u}, \hat{w}_N)$  updated using  $(\bar{u}, \bar{w}_N)$  is

$$\hat{u}^\top = \bar{u} + \alpha_D (z^{i^*})^\top \quad (5.28)$$

$$\hat{w}_N^\top = \bar{w}_N^\top - \alpha_D (z^{i^*})^\top A_N \quad (5.29)$$

$$\hat{w}_B^\top = 0. \quad (5.30)$$

The simplex algorithm is generating a direction vector for the dual vector  $u$  which is row  $i^*$  of  $A_B^{-1}$ . The dual step size is  $\alpha_D$ . The subscript of  $P$  or  $D$  on  $\alpha$  indicates the step size in either primal or dual space.

The update formulas (5.28)-(5.29) are equivalent to the calculations

$$\hat{u}^\top = c_B^\top A_B^{-1}, \quad \hat{w}_N^\top = c_N^\top - \hat{u}^\top A_N. \quad (5.31)$$

However,  $z^{i^*}$  is usually a sparser vector than  $\hat{u}$  and (5.28)-(5.29) can be implemented more efficiently than (5.31). This issue is discussed further in Chapter 13.

**Algorithm 5.9 (Revised Simplex)**

**Step 1: (Initialization)** Given a basis matrix  $A_B$  such that  $A_B^{-1}b \geq 0$ , initialize the primal and dual solutions

$$\bar{x}_B \leftarrow A_B^{-1}b, \bar{z}_0 \leftarrow c_B^\top A_B^{-1}b, \bar{u}^\top \leftarrow c_B^\top A_B^{-1}, \bar{w}_N^\top \leftarrow c_N^\top - \bar{u}^\top A_N.$$

**Step 2: (Pivot Column Selection)** If  $\bar{w}_N \geq 0$ , stop, the current solution is optimal; else, select  $q \in N$  such that  $\bar{w}_q < 0$ .

**Step 3: (Update the Pivot Column)** Calculate the updated column  $\bar{a}^q$

$$\bar{a}^q \leftarrow A_B^{-1}a^q.$$

**Step 4: (Primal Minimum Ratio Test and Pivot Row Selection)** If  $\bar{a}_{iq} \leq 0$  for  $i = 1, \dots, m$  stop, the linear program is unbounded. Otherwise, perform the primal minimum ratio test

$$i^* \leftarrow \operatorname{argmin} \{\bar{x}_{B_i}/\bar{a}_{iq} \mid \bar{a}_{iq} > 0, i = 1, \dots, m\}$$

Update the primal and dual step sizes.

$$\alpha_P \leftarrow \bar{x}_{B_{i^*}}/\bar{a}_{i^*q}, \quad \alpha_D \leftarrow \bar{w}_q/\bar{a}_{i^*q}$$

**Step 5: (Row Update)** Find row  $i^*$  of  $A_B^{-1}$  by

$$(z^{i^*})^\top \leftarrow (e^{i^*})^\top A_B^{-1}.$$

**Step 6: (Update Primal and Dual Solution, Basis and Basis Inverse)**

$$\bar{x}_{B_i} \leftarrow \bar{x}_{B_i} - \alpha_P \bar{a}_{iq}, \quad i = 1, \dots, m, \quad \bar{x}_q \leftarrow \alpha_P, \quad \bar{z}_0 \leftarrow \bar{z}_0 + \alpha_P \bar{w}_q$$

$$\bar{u}^\top \leftarrow \bar{u} + \alpha_D (z^{i^*})^\top, \quad \bar{w}_N^\top \leftarrow \bar{w}_N^\top - \alpha_D (z^{i^*})^\top A_N, \quad \bar{w}_{B_{i^*}} \leftarrow -\alpha_D$$

Update the index of basic variables.

$$B \leftarrow (B \setminus \{B_{i^*}\}) \cup \{q\}, \quad N \leftarrow (N \setminus \{q\}) \cup \{B_{i^*}\}$$

and update the basis inverse

$$A_B^{-1} \leftarrow \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*q}} \right) A_B^{-1}.$$

Go to Step 2.

**Example 5.10 (Example 5.2 continued)****Iteration 1**

**Step 1: (Initialization)** The initial basis is  $B = \{1, 2, 5, 6\}$  and the nonbasic variable indices are  $N = \{3, 4\}$ . The initial values of the dual variables are

$$\bar{u}^T = c_B^T A_B^{-1} = [2 \ -3 \ 0 \ 0] \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1.5 & .5 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} = [-5 \ 2 \ 0 \ 0]$$

The initial reduced costs are

$$\bar{w}_N^T = c_N^T - \bar{u}^T A_N = [0 \ 0] - [-5 \ 2 \ 0 \ 0] \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = [-5 \ 2]$$

**Step 2: (Pivot Column Selection)** Variable  $x_3$  is the only variable with a negative reduced cost, select it to enter the basis.

**Step 3: (Update the Pivot Column)** Calculate

$$\bar{a}^q = A_B^{-1} a^3 = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1.5 & .5 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1.5 \\ 2 \end{bmatrix}$$

**Step 4: (Primal Minimum Ratio Test and Pivot Row Selection)** As seen previously,  $i^* = 1$  and variable  $x_1$  leaves the basis. At this iteration,  $\bar{a}_{13} = 1$ ,  $\bar{x}_1 = 1$  and  $\bar{w}_3 = -5$  so the primal and dual step lengths are  $\alpha_P = 1$  and  $\alpha_D = -5$ .

**Step 5: (Row Update)** Find row 1 of  $A_B^{-1}$ .

$$z^1 = (e^1)^T A_B^{-1} = [1 \ 0 \ 0 \ 0] \begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -1.5 & .5 & 1 & 0 \\ -2 & 1 & 0 & 1 \end{bmatrix} = [-1 \ 1 \ 0 \ 0].$$

**Step 6: (Update Primal and Dual solutions, Basis and Basis Inverse)**

The new primal solution is

$$\begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_5 \\ \bar{x}_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 7.5 \\ 6 \end{bmatrix} - 1 \begin{bmatrix} 1 \\ -1 \\ 1.5 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 6 \\ 4 \end{bmatrix}$$

and  $\bar{x}_3 = 1$ .

The updated dual variables using (5.28)-(5.29) are

$$\begin{aligned} \hat{w}^T &= \bar{w}^T + \alpha_D(z^1)^T \\ &= [-5 \ 2 \ 0 \ 0] - 5[-1 \ 1 \ 0 \ 0] = [0 \ -3 \ 0 \ 0] \end{aligned}$$

$$\begin{aligned} \hat{w}_{\hat{N}}^T &= \bar{w}_{\hat{N}}^T - \alpha_D(z^1)^T A_{\hat{N}} \\ &= [0 \ 2] + 5[-1 \ 1 \ 0 \ 0] \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = [5 \ -3] \end{aligned}$$

The new basis is  $\hat{B} = \{3, 2, 5, 6\}$  and the new inverse is  $EA_B^{-1}$  where

$$E = I - \frac{(\bar{a}^3 - e^1)(e^1)^T}{\bar{a}_{13}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -1.5 & 0 & 1 & 0 \\ -2 & 0 & 0 & 1 \end{bmatrix}.$$

It is instructive to compare the complexity of an iteration of the revised simplex algorithm with the simplex algorithm as described in the previous section. In this analysis, we assume that the  $A$  matrix is sparse. Indeed, if  $A$  were not sparse it would be impossible to solve large problems. Assume that on average there are  $h$  nonzeros per column in the  $A$  matrix. This sparsity assumption is crucial in taking the inner product of a sparse vector with a dense vector. By properly taking advantage of sparsity, the inner product of an  $m$  dimensional vector with  $h$  elements nonzero with a dense  $m$  dimensional vector requires  $h$  multiplications and  $h-1$  additions. We discuss sparse inner products in greater detail in Chapter 13. In Table 5.4 we assume that each column of  $A$  contains  $h$  nonzero elements and that the columns of  $A_B^{-1}$  and  $A_B^{-1}A_N$  are dense, i.e. contain  $m$  nonzero elements and that column  $i^*$  of  $E$  has  $m$  nonzero elements. In this table we give a rough comparison of the multiplications required in

**Table 5.1** Multiplications Required for One Iteration of Revised Simplex

Operation	Standard Simplex	Revised Simplex
Pivot Column Update $A_B^{-1}a^q$	0	$mh$
Update $\bar{x}_B$	0	$m$
Update $\bar{w}_N$	0	$(n - m)h + m$
Basis Update	0	$m^2$
Pivot	$(n - m)m$	0

the standard simplex algorithm using a full tableau with the revised simplex algorithm.

In the dual update calculation, assume  $z^{i^*}$  is dense and so the operation  $\alpha_D z^{i^*}$  is  $m$  multiplications and then this dense vector is multiplied by the  $n - m$  nonbasic columns with  $h$  nonzero elements per column. In the pivot calculation for standard simplex, we assume that updating the nonbasic columns in the tableau requires approximately  $(n - m)m$  multiplications. In many applications,  $n \gg m$ , and  $m \gg h$  so the  $nm$  term in standard simplex is substantially larger than the  $nh$  and  $m^2$  terms for revised simplex. A similar analysis applies to the number of additions. If  $m \approx h$  revised simplex requires more work than standard simplex.

## 5.5 PRODUCT FORM OF THE INVERSE

In the revised simplex algorithm,  $A_B^{-1}$  is required in two steps. First, the basis inverse is necessary for calculating  $\bar{a}^q = A_B^{-1}a^q$  which is required in the minimum ratio test, and second, for calculating  $(z^{i^*})^\top = (e^{i^*})^\top A_B^{-1}$  which is required for updating the dual solution. Since these calculations are made at every pivot, it is important that they be done in a very efficient manner. It is not practical to explicitly calculate and store  $A_B^{-1}$  because even if  $B$  is very sparse,  $A_B^{-1}$  is often very dense. Consider the following example matrix from Chvátal [93], page 96.

### Example 5.11

$$\left[ \begin{array}{ccccc} 1 & & & & 1 \\ 1 & 1 & & & \\ 1 & 1 & 1 & & \\ & & 1 & 1 & \\ & & & 1 & 1 \end{array} \right]$$

This matrix has only two nonzero elements per column yet the inverse of this matrix is completely dense.

In order to take advantage of sparsity of the basis matrix  $A_B$ , the inverse  $A_B^{-1}$  is stored in *product form*. The product form of the inverse is based on the rank one update formula developed earlier. If  $\hat{B} = B \setminus \{B_{i^*}\} \cup \{q\}$ , then from Lemma 5.8,  $A_{\hat{B}}^{-1} = EA_B^{-1}$  where

$$E = I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*q}}.$$

Define the  $m$  component *eta* vector  $\eta$  by

$$\eta := \begin{cases} -\bar{a}_{iq}/\bar{a}_{i^*q}, & i \neq i^*, \\ 1/\bar{a}_{i^*q}, & i = i^*. \end{cases} \quad (5.32)$$

Then  $E$  is

$$E = I + (\eta - e^{i^*})(e^{i^*})^\top = [ e^1 \ e^2 \ \dots \ e^{i^*-1} \ \eta \ e^{i^*+1} \ \dots \ e^m ].$$

The matrix  $E$  is called an *eta matrix* or an *elementary matrix*. If  $y$  is an arbitrary  $m$  component vector then

$$Ey = \begin{bmatrix} y_1 - y_{i^*}(\bar{a}_{1q}/\bar{a}_{i^*q}) \\ y_2 - y_{i^*}(\bar{a}_{2q}/\bar{a}_{i^*q}) \\ \vdots \\ y_{i^*}/\bar{a}_{i^*q} \\ \vdots \\ y_m - y_{i^*}(\bar{a}_{mq}/\bar{a}_{i^*q}) \end{bmatrix}$$

The effect of pre-multiplying a column vector by the elementary matrix  $E$  has the effect of carrying out the pivot operations on that column. Because of the special structure  $E$  has, it is not necessary to store the entire matrix. It is only necessary to store the eta vector and the integer  $i^*$  corresponding to the pivot row. Thus, the basis matrix at any iteration is written as an elementary matrix times the basis inverse from the previous iteration. If the initial basis matrix is written as a product of elementary matrices then at every iteration the basis is written as the product of elementary matrices. We illustrate writing the initial basis in product form for Example 5.2.

**Example 5.12 (Example 5.2 continued)** The initial basis matrix for Example 5.2 is

$$A_B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -.5 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}.$$

The textbook method for calculating  $A_B^{-1}$  is to append an identity matrix to  $A_B$  and perform the same row exchanges and pivots on the identity as on  $A_B$ . The first step in inverting  $A_B$  is to exchange the first and second row in order to get a nonzero in element in the first column of the first row. This corresponds to pre-multiplying  $A_B$  by the identity matrix with rows one and two interchanged. Call this permutation matrix  $P_{12}$ .

$$P_{12}A_B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -.5 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}$$

The next step in inverting  $A_B$  is to pivot on the element in column one, row one. This involves multiplying row one by .5 and adding the result to row three and then adding row one to row four. This corresponds to pre-multiplying  $P_{12}A_B$  by the elementary matrix

$$E_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ .5 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Then

$$E_1 P_{12} A_B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1.5 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

Now pivot on element (2, 2). This corresponds to pre-multiplying by

$$E_2 = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1.5 & 1 & 0 \\ 0 & -2 & 0 & 1 \end{bmatrix}$$

Then  $A_B^{-1} = E_2 E_1 P_{12}$  since

$$E_2 E_1 P_{12} A_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which is an identity matrix. Then the inverse of a nonsingular basis matrix  $A_B$ , at an arbitrary iteration of the revised simplex algorithm, (assuming, for simplicity, no row exchanges) is the product of  $k$  elementary matrices, that is

$$A_B^{-1} = E_k E_{k-1} \cdots E_1. \quad (5.33)$$

Before continuing, consider again the matrix of Example 5.11 used to introduce this section. The product form of the inverse of this matrix is very sparse. In general, the product form of the inverse of a matrix with this structure of size  $2n+1 \times 2n+1$  requires storing  $2n+1$  eta vectors with a total of  $6n+1$  nonzero elements (not including the row index  $i^*$ ). If the inverse is stored explicitly,  $(2n+1)^2$  nonzero elements are stored.

The product form of the inverse (5.33) is used to calculate  $\bar{a}^q$  and  $z^{i^*}$  as follows.

$$\bar{a}^q = A_B^{-1} a^q = E_k E_{k-1} \cdots E_1 a^q \quad (5.34)$$

$$(z^{i^*})^\top = (e^{i^*})^\top A_B^{-1} = (e^{i^*})^\top E_k E_{k-1} \cdots E_1 \quad (5.35)$$

The sequence of matrix multiplications  $E_k E_{k-1} \cdots E_1 a^q$  in (5.34) is known as an FTRAN and the sequence of matrix multiplications  $(e^{i^*})^\top E_k E_{k-1} \cdots E_1$  in (5.35) is known as a BTRAN. These terms originated in the early days of linear programming when the basis inverse was not stored in core, rather it was stored on tape. In the FTRAN operation the tape is run forward to calculate  $\bar{a}^q$  and backward in the BTRAN to calculate  $z^{i^*}$ .

In making the FTRAN and BTRAN multiplications, a full matrix multiplication is not necessary. Let  $y$  be an arbitrary  $m$  vector. Consider the FTRAN operation. If  $E_l$  is an elementary matrix corresponding to a pivot on element  $\bar{a}_{i^*q}$  then

$$E_l y = y + (\eta^l - e^{i^*})(e^{i^*})^\top y$$

This calculation is made by multiplying the scalar  $y_{i^*}$  times the eta vector  $\eta^l$  and adding the result to  $y$  except for component  $i^*$  which is replaced by  $\eta_{i^*} y_{i^*}$ .

An arbitrary BTRAN operation is

$$y^T E_i = y^T + (y^T \eta^i - y^T e^{i*})(e^{i*})^T$$

This calculation is made by calculating the inner product  $y^T \eta^i$  and then replacing component  $i^*$  of  $y$  with the result of the inner product.

**Example 5.13 (Example 5.2 continued)** The initial basis is  $B = \{1, 2, 5, 6\}$ . At the first iteration of revised simplex variable  $x_3$  was selected to enter the basis and by the minimum ratio test variable  $x_1$  was selected to leave the basis. The updated column is

$$\bar{a}^3 = \begin{bmatrix} 1 \\ -1 \\ 1.5 \\ 2 \end{bmatrix}.$$

Since  $x_1$  leaves the basis and  $x_1$  is basic in row one, the first element of  $\bar{a}^3$  is the pivot element. Using formula (5.32) for calculating the eta vector gives

$$\eta^3 = \begin{bmatrix} 1 \\ 1 \\ -1.5 \\ -2 \end{bmatrix}.$$

After updating the dual variables for the new basis  $\hat{B} = \{3, 2, 5, 6\}$  using (5.28)-(5.29), variable  $x_4$  is the only nonbasic variable with a negative reduced cost and it is selected to enter the basis in the second pivot. In order to perform the minimum ratio test in Step 4 it is necessary to calculate  $\bar{a}^4 = A_{\hat{B}}^{-1} a^4$ . We illustrate this calculation using an FTRAN with the product form of the inverse. The eta file is used to calculate the updated column  $\bar{a}^4$  using an FTRAN. At this point the eta file is the permutation matrix  $P_{12}$  and the eta vectors

$$\eta^1 = \begin{bmatrix} 1 \\ 0 \\ .5 \\ 1 \end{bmatrix}, \quad \eta^2 = \begin{bmatrix} -1 \\ 1 \\ -1.5 \\ -2 \end{bmatrix}, \quad \eta^3 = \begin{bmatrix} 1 \\ 1 \\ -1.5 \\ -2 \end{bmatrix}.$$

The FTRAN is  $\bar{a}^4 = E_3 E_2 E_1 P_{12} a^4$ . This calculation is made as follows.

$$\bar{a}^4 \leftarrow P_{12} a^4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\bar{a}^4 \leftarrow E_1 \bar{a}^4 = \bar{a}^4 + (\eta_1 - e^1)(e^1)^T \bar{a}^4 = \begin{bmatrix} -1 \\ 0 \\ -\frac{1}{2} \\ -1 \end{bmatrix}$$

$$\bar{a}^4 \leftarrow E_2 \bar{a}^4 = \bar{a}^4 + (\eta_2 - e^2)(e^2)^T \bar{a}^4 = \begin{bmatrix} -1 \\ 0 \\ -\frac{1}{2} \\ -1 \end{bmatrix}$$

$$\bar{a}^4 \leftarrow E_3 \bar{a}^4 = \bar{a}^4 + (\eta_3 - e^1)(e^1)^T \bar{a}^4 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

From the minimum ratio test variable  $x_6$  is selected to leave the basis. It is currently basic in row 4 so it is necessary to calculate  $(z^4)^T = (e^4)^T A_B^{-1}$  using a BTRAN operation. The BTRAN sequence is:

$$\begin{aligned} (z^4)^T &\leftarrow (e^4)^T E_3 = (e^4)^T + ((e^4)^T \eta_3 - (e^4)^T e^1)(e^1)^T = [-2 \ 0 \ 0 \ 1] \\ (z^4)^T &\leftarrow (z^4)^T E_2 = (z^4)^T + ((z^4)^T \eta_2 - (z^4)^T e^2)(e^2)^T = [-2 \ 0 \ 0 \ 1] \\ (z^4)^T &\leftarrow (z^4)^T E_1 = (z^4)^T + ((z^4)^T \eta_1 - (z^4)^T e^1)(e^1)^T = [-1 \ 0 \ 0 \ 1] \\ (z^4)^T &\leftarrow (z^4)^T P_{12} = [0 \ -1 \ 0 \ 1]. \end{aligned}$$

If the eta file becomes too long, it is more efficient to delete the current eta file and reinvert the current basis matrix from scratch. Consider the number of multiplications during the FTRAN and BTRAN operation. If each eta vector has, on average,  $d$  nonzero elements, then each eta vector in the file contributes approximately  $d$  multiplications to the FTRAN and  $d$  multiplications during to the BTRAN. Let  $k$  be the number of iterations between consecutive matrix reinversions from scratch. Assume that factoring the basis from scratch results in  $m$  eta vectors. If each of these eta vector has approximately  $d$  nonzero elements, then the reinversion requires approximately  $(m^2/2)d$  multiplications. Then  $2md$  multiplications are required for the first iteration (BTRAN and FTRAN) not including the work of reinverting,  $2md + 2d$  for the second iteration, and  $2(m+k-1)d$  for iteration  $k$ . The total number of multiplications for the cycle including reinverting the matrix at the beginning of the cycle is  $(m^2/2)d + 2mdk + k(k-1)md$ . Then the average work per iteration is  $(m^2d/2k) + 2md + (k-1)d$ . Taking the first derivative with respect to  $k$ , setting to zero and solving gives  $k = (m/\sqrt{2})$  as the optimum number of iterations per cycle. In this calculation we have ignored the issue numerical accuracy and it may be necessary to reinvert more often in order to preserve numerical stability.

of the problem. Often codes such as LINDO have simple reinversion rules such as to reinvert every 200 pivots or reinvert if the number of nonzero elements in the eta file is more than twice the number of nonzero elements in the constraint matrix.

In summary, with the product form of the inverse, the only change in revised simplex is how  $A_B^{-1}$  is stored. In Step 6 of the revised simplex using the product form of the inverse, the inverse is stored as a product of eta vectors and one new eta vector is added at each pivot rather than multiplying the new eta matrix times  $A_B^{-1}$ . Storing the  $A_B^{-1}$  explicitly requires  $O(m^2)$  words even if  $A_B$  is sparse. In general, we expect  $d$ , the number of nonzeros in the eta vectors to be much less than  $m$ , so using the product form of the inverse greatly reduces the memory requirements over using an explicit  $A_B^{-1}$ . For example, if the basis is nearly triangular then eta vectors will have almost the same density as the  $A$  matrix. Again, consider Example 5.11.

In Table 5.2 we compare the multiplications required for key steps of the revised simplex algorithm with the explicit form of the inverse to the revised simplex method with product form of the inverse. Assume that there are  $l$  of vectors in the eta file each with  $d$  nonzero elements, each column of  $A_N$  has  $h$  nonzeros and that  $A_B^{-1}$  is 100% dense. This table probably overstates the work required

**Table 5.2** Multiplications Required for One Iteration of Revised Simplex with Explicit and Product form of Inverse

Operation	Revised Simplex Explicit Inverse	Revised Simplex Product Form of Inverse
Pivot Column Update $A_B^{-1}a^q$	$mh$	$ld$ (FTRAN)
Find $z^i$ *	0	$ld$ (BTRAN)
Update $\bar{w}_N$	$(n - m)h + m$	$(n - m)h + m$
Inverse Update	$md$	0

for the revised simplex with the product form of the inverse. We leave it as an exercise to explain why this is true. The product form of the inverse was used for many years until researchers began using the *LU* decomposition of the basis and updating it at each pivot. This is described Chapter 13.

## 5.6 DEGENERACY AND CYCLING

The proof of convergence for the simplex algorithm presented earlier assumed that there were no degenerate solutions associated with any tableau. The

problem caused by degeneracy is that if a basic feasible solution is degenerate at least one  $\bar{x}_{B_i}$  is zero. If a variable selected to pivot into the basis has a positive coefficient in row  $i$  then the minimum ratio is zero and pivoting the variable into the basis does not result in a strict decrease in the objective function. This implies there is a possibility that the simplex algorithm will *cycle*. That is, a basic feasible solution is repeated during pivoting. This corresponds to an “infinite loop” and the simplex algorithm will not converge in a finite number of pivots. An example of this behavior is given by Beale [45].

**Example 5.14 (Beale)** *This example is due to Beale [45].*

$$\begin{array}{rccccccccl} z_0 & +\frac{3}{4}x_4 & -20x_5 & +\frac{1}{2}x_6 & -6x_7 & = & 0 \\ x_1 & +\frac{1}{4}x_4 & -8x_5 & -x_6 & +9x_7 & = & 0 \\ x_2 & +\frac{1}{2}x_4 & -12x_5 & -\frac{1}{2}x_6 & +3x_7 & = & 0 \\ x_3 & & & & +x_6 & & = & 1 \end{array}$$

Pivot  $x_4$  into the basis and pivot  $x_1$  out of the basis.

$$\begin{array}{rccccccccl} z_0 & -3x_1 & +4x_5 & +\frac{7}{2}x_6 & -33x_7 & = & 0 \\ x_4 & +4x_1 & -32x_5 & -4x_6 & +36x_7 & = & 0 \\ x_2 & -2x_1 & +4x_5 & +\frac{3}{2}x_6 & -3x_7 & = & 0 \\ x_3 & & +x_6 & & & & = & 1 \end{array}$$

Pivot  $x_5$  into the basis and  $x_2$  out of the basis.

$$\begin{array}{rccccccccl} z_0 & -x_1 & -x_2 & +2x_6 & -18x_7 & = & 0 \\ x_4 & -12x_1 & +8x_2 & +8x_6 & -84x_7 & = & 0 \\ x_5 & -\frac{1}{2}x_1 & +\frac{1}{4}x_2 & +\frac{3}{8}x_6 & -\frac{15}{4}x_7 & = & 0 \\ x_3 & & +x_6 & & & & = & 1 \end{array}$$

Pivot  $x_6$  into the basis and  $x_4$  out of the basis.

$$\begin{array}{rccccccccl} z_0 & +2x_1 & -3x_2 & -\frac{1}{4}x_4 & +3x_7 & = & 0 \\ x_6 & -\frac{3}{2}x_1 & +x_2 & +\frac{1}{8}x_4 & -\frac{21}{4}x_7 & = & 0 \\ x_5 & +\frac{1}{16}x_1 & -\frac{1}{8}x_2 & -\frac{2}{64}x_4 & +\frac{3}{16}x_7 & = & 0 \\ x_3 & +\frac{9}{2}x_1 & -x_2 & -\frac{8}{8}x_4 & +\frac{16}{2}x_7 & = & 1 \end{array}$$

Pivot  $x_7$  into the basis and  $x_5$  out of the basis.

$$\begin{array}{rccccccccl} z_0 & +x_1 & -x_2 & +\frac{1}{4}x_4 & -16x_5 & = & 0 \\ x_6 & +2x_1 & -6x_2 & -\frac{5}{2}x_4 & +56x_5 & = & 0 \\ x_7 & +\frac{1}{3}x_1 & -\frac{2}{3}x_2 & -\frac{1}{4}x_4 & +\frac{16}{3}x_5 & = & 0 \\ x_3 & -2x_1 & +6x_2 & +\frac{5}{2}x_4 & -56x_7 & = & 1 \end{array}$$

Pivot  $x_1$  into the basis and  $x_6$  out of the basis.

$$\begin{array}{ccccccc} z_0 & -\frac{1}{2}x_6 & +2x_2 & +\frac{7}{4}x_4 & -44x_5 & = & 0 \\ x_1 & +\frac{1}{2}x_6 & -3x_2 & -\frac{5}{4}x_4 & +28x_5 & = & 0 \\ x_7 & -\frac{1}{6}x_6 & +\frac{1}{3}x_2 & +\frac{1}{6}x_4 & -4x_5 & = & 0 \\ x_3 & +x_6 & & & & = & 1 \end{array}$$

Pivot  $x_2$  into the basis and pivot  $x_7$  out of the basis.

$$\begin{array}{ccccccc} z_0 & +\frac{1}{2}x_6 & -6x_7 & +\frac{3}{4}x_4 & -20x_5 & = & 0 \\ x_1 & +x_6 & +9x_7 & +\frac{1}{4}x_4 & -8x_5 & = & 0 \\ x_2 & -\frac{1}{2}x_6 & +3x_7 & +\frac{1}{2}x_4 & -12x_5 & = & 0 \\ x_3 & +x_6 & & & & = & 1 \end{array}$$

This is identical to the first tableau and we have an infinite loop.

In this section we give a rule for selecting the entering and the leaving variable that prevents cycling. Here is the basic idea. Assume that at iteration  $t$  the basic feasible solution associated with the basis matrix  $B^t$  is degenerate and that the variables and rows are ordered such that  $\bar{x}_{B_i^t} = 0$ , for  $i = 1, \dots, k$ , and  $\bar{x}_{B_i^t} > 0$ , for  $i = k+1, \dots, m$ . The nonbasic variables are indexed by  $N^t$ . In this section, we use a superscript on  $B$  to indicate the set of basic variables at the start of iteration  $t$ . Similarly for the nonbasic variables. Rather than use an overline, we use  $a_{ij}^t$  to denote the elements of  $A_{B^t}^{-1} A_{N^t}$  at the start of iteration  $t$ . Similarly, for the reduced cost  $w_j^t$ . The problem of finding a variable which leads to a strict decrease in the objective function is equivalent to finding an extreme ray  $\bar{x}$  of the *degeneracy cone* defined by the constraints

$$x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j = 0, \quad i = 1, \dots, k \quad (5.36)$$

$$x_i \geq 0, \quad i = 1, \dots, n \quad (5.37)$$

such that  $c^T \bar{x} < 0$ . That is, perform a sequence  $t+1, \dots, r$  of simplex pivots on this cone until there is a column  $l \in N^r$  in the tableau

$$x_{B_i^r} + \sum_{j \in N^r} a_{ij}^r x_j = 0, \quad i = 1, \dots, k, \quad (5.38)$$

where  $w_l^r < 0$  and  $a_{il}^r \leq 0$ ,  $i = 1, \dots, k$ . If  $a_{il}^r \leq 0$ ,  $i = 1, \dots, m$  then the original linear program is unbounded. Otherwise, there is an  $i > k$  such that  $a_{il}^r > 0$  and the minimum ratio for variable  $x_l$  is positive. Pivoting on variable  $l$  results in a strict decrease in the objective function. We need a simplex implementation, that starting with the tableau (5.36) generates the tableau (5.38) where  $w_j^r \geq 0$

for all  $j \in N^r$ , in which case we have an optimal solution for the linear program; or  $w_l^r < 0$  and  $a_{il}^r \leq 0$ ,  $i = 1, \dots, k$  and either a non-degenerate pivot is made or the problem is unbounded.

L. Schrage<sup>1</sup> has observed that if a variable  $x_{B_i^t}$  is fixed temporarily and not allowed to exit the basis then the problem of finding an extreme ray  $\bar{x}$  for the  $k$  constraint degeneracy cone (5.36)-(5.37) reduces to finding an extreme ray of the  $(k - 1)$  constraint degeneracy cone

$$x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j = 0, \quad i = 2, \dots, k \quad (5.39)$$

$$x_i \geq 0, \quad i = 1, \dots, n \quad (5.40)$$

which is derived from the cone (5.36)-(5.37) by deleting the first row. If at an iteration  $r \geq t$  there is a variable  $l \in N^r$  such that  $w_l^r < 0$  and  $a_{il}^r \leq 0$ ,  $i = 2, \dots, k$ , then either  $a_{1l}^r \leq 0$ , in which case we have an extreme ray of the cone (5.36) - (5.37) with negative objective function value; or,  $a_{1l}^r > 0$  and it is necessary to pivot out variable  $x_{B_1^r}$ . After pivoting out variable  $x_{B_1^r}$ , this nonbasic column has a positive reduced cost, and nonnegative elements in rows 1 through  $k$  in the new tableau. By Lemma 5.15 below (which is left as an exercise), if there is an extreme ray of the degeneracy cone with negative cost, there is an extreme ray of the degeneracy cone with negative cost and  $x_{B_1^t} = 0$ . This implies that this variable never needs to be considered again and is deleted from the list of potential entering variables. Thus, variable  $x_{B_1^t}$  can never be part of a cycle.

**Lemma 5.15** *If there exists an extreme ray  $\bar{x}$  of the cone defined by (5.36) - (5.37) such that  $c^\top \bar{x} < 0$  and if at some pivot  $r \geq t$ , there is a variable  $l \in N^r$  with the property that  $w_l^r \geq 0$ ,  $a_{il}^r \geq 0$ ,  $i = 1, \dots, k$ , then there is an extreme ray  $\hat{x}$  of this cone such that  $c^\top \hat{x} < 0$  and  $\hat{x}_l = 0$ .*

Recursively applying this idea provides a convergent pivoting scheme. The following rules for selecting the entering and exiting variables are due to Cacioppi and Schrage [77].

**Cacioppi and Schrage Rule:** Apply the simplex algorithm to the degeneracy cone (5.36)-(5.37).

1. **Initialization:** Initialize basic variable  $x_{B_i^t}$  to level  $i$ ,  $i = 1, \dots, m$ . Initialize all nonbasic variables to level  $\infty$ .

---

<sup>1</sup>Personal conversation, November, 1995.

2. *Entering Variable:* Select an entering variable with negative reduced cost from among all level  $\infty$  nonbasic variables. Any variable with negative reduced cost is a valid choice. If there are no level  $\infty$  nonbasic variables with a negative reduced cost, terminate because there is no ray of the cone with a negative cost. The validity of this is shown in Lemma 5.18.
3. *Exit Variable:* If there are ties by the minimum ratio test among leaving variables, pivot out the basic variable  $x_{B_i^t}$  with largest index  $i$ . When  $x_{B_i^t}$  is pivoted out of the basis and becomes nonbasic assign it level  $i - 1$  and reassign all variables with level  $i$  or greater, to level  $\infty$ .

The idea behind the exit rule is a recursive application of Lemma 5.15. Assume variable  $x_{B_i^t}$  exits the basis at pivot  $t$ . If there are no more pivots in rows  $1, \dots, i - 1$  of the degeneracy cone then apply Lemma 5.15 to the degeneracy cone with rows  $i, \dots, k$  and correctly conclude that variable  $x_{B_i^t}$  is not positive in any negative cost extreme ray of the reduced degeneracy cone. By giving this variable a level of  $i - 1$  and allowing only variables with level  $\infty$  to enter the basis, variable  $x_{B_i^t}$  is not eligible to enter the basis again as long as all pivots are done in rows  $i, \dots, k$ . However, if there is a pivot in row  $i - 1$  or less, then such a pivot can affect rows  $i, \dots, k$  so it is necessary to consider this variable once again for entry into the basis. If there is a pivot in row  $t < i$ , then the exit rule will reset variable  $x_{B_i^t}$  to have a level of  $\infty$ .

**Lemma 5.16** *If the Cacioppi and Schrage Rule is applied to a degenerate tableau (5.36)-(5.37) and basic variable  $x_{B_i^t}$  pivots out of the basis, then if there is an extreme ray of the cone defined by (5.36)-(5.37), with negative cost there is an extreme ray of this cone with negative cost and variable  $x_{B_i^t}$  fixed to zero.*

**Proof:** If variable  $x_{B_i^t}$  pivots out of the basis, then by the exit variable rule, none of the basic variable  $x_{B_i^t}$  for  $i = 2, \dots, k$  were eligible. If nonbasic variable  $l$  was selected as the entering variable this implies  $a_{il}^t \leq 0$  for  $i = 2, \dots, k$ . Then  $a_{iB_i^t}^{t+1} \geq 0$  for  $i = 1, \dots, k$  and  $c_{B_i^t}^{t+1} > 0$ . The result now follows from Lemma 5.15.  $\square$

**Lemma 5.17** *Assume the simplex algorithm applied to the linear program*

$$\min \quad \sum_{j \in N^t} w_j^t x_j$$

$$\begin{aligned} \text{s.t. } & x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j = \bar{x}_{B_i^t}, \quad i = 1, \dots, k \\ & x_{B_i^t} \geq 0, \quad i = 1, \dots, k \\ & x_j \geq 0, \quad j \in N^t \end{aligned}$$

finds a primal-dual optimal solution. If none of the simplex pivots occur in rows,  $i = 1, \dots, l-1 < k$ , then given an additional column  $h \notin N^t$  such that  $a_{ih} \geq 0$ ,  $i = l, \dots, k$  and  $c_h > 0$ , this column prices out with a nonnegative reduced cost.

**Proof:** By hypothesis, simplex finds an optimal primal-dual solution. Let  $u_i$  denote the dual variables on the constraints

$$x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j = \bar{x}_{B_i^t},$$

$w_{B_i^t}$  the dual variables on the  $x_{B_i^t} \geq 0$  constraints and  $w_j$  the dual variables on the  $x_j \geq 0$ ,  $j \in N^t$  constraints. The dual problem is

$$\begin{aligned} \max \quad & \sum_{i=1}^k \bar{x}_{B_i^t} u_i \\ \text{s.t.} \quad & u_i + w_{B_i^t} = 0, \quad i = 1, \dots, k \\ & \sum_{i=1}^k u_i a_{ij}^t + w_j = w_j^t, \quad j \in N^t \\ & w_{B_i^t} \geq 0, \quad i = 1, \dots, k \\ & w_j \geq 0, \quad j \in N^t. \end{aligned}$$

By hypothesis, the simplex method terminates at some pivot  $r \geq t$  with an optimal primal-dual solution. Let  $B^r$  denote the optimal basis when simplex terminates. By hypothesis, no pivots occur in rows  $i = 1, \dots, l-1$ . This implies that  $w_i^r = 0$  for  $i = 1, \dots, l-1$ , and columns  $i = 1, \dots, l-1$  of  $A_{B^r}^{-1}$  are the unit vectors  $e^i$ . Let  $\bar{u}$  be the optimal dual variables upon simplex termination. Then  $\bar{u}_i = 0$  for  $i = 1, \dots, l-1$  since the objective function coefficients of the basic variables are 0. The dual constraints  $u_i + w_{B_i^r} = 0$  and  $w_{B_i^r} \geq 0$  imply  $\bar{u}_i \leq 0$  for  $i = l, \dots, k$ . Therefore, if a new column  $h$  is added to the linear program with  $a_{ih} \geq 0$  for  $i = l, \dots, k$  and  $c_h > 0$ , then the reduced cost on this new column is

$$c_h - \sum_{i=1}^k \bar{u}_i a_{ih} > 0$$

which is strictly positive.  $\square$

**Lemma 5.18** *If the simplex algorithm applied to the degeneracy cone*

$$\begin{aligned} \min \quad & \sum_{j \in N^t} w_j^t x_j \\ \text{s.t.} \quad & x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j = 0, \quad i = 1, \dots, k \\ & x_{B_i^t} \geq 0 \quad i = 1, \dots, k \\ & x_j \geq 0 \quad j \in N^t \end{aligned}$$

*using the Cacioppi and Schrage entering and exit variable rules terminates because all nonbasic variables with level  $\infty$  have nonnegative reduced costs, then there is no extreme ray of the degeneracy cone with a negative cost and all variables assigned a level 0 to  $k - 1$  also have a nonnegative reduced cost.*

**Proof:** It suffices to show upon termination that all variables assigned a level 0 to  $k - 1$  have a nonnegative reduced cost since this implies there is an optimal solution to the degeneracy cone which implies there is no extreme ray with a negative cost. Consider a variable  $x_h$  which has a level  $l - 1 \leq k - 1$ . If all of the level  $\infty$  variables have a nonnegative reduced costs, then there was a pivot  $r$  such that variable  $x_h$  was selected as the exit variable. After pivot  $r$ , there was never a pivot in rows  $i = 1, \dots, l - 1$ , otherwise  $x_h$  would be assigned a level of  $\infty$ . Since  $x_h$  has a level of  $l - 1$  it was pivoted out of row  $l$ . This implies that the entering variable has its first positive element in row  $l$ . Therefore, after  $x_h$  exits the basis, in the updated tableau it has nonnegative entries in rows  $l$  through  $k$ . Because this variable has a level less than infinity it is never again selected to enter the basis and the simplex algorithm operates on the tableau as if this variable were not present. The conditions of Lemma 5.17 are met and we conclude that this variable has a nonnegative reduced cost.  $\square$

The following Lemma is left as an exercise.

**Lemma 5.19** *Consider the cone*

$$C = \{x \in \mathbb{R}^n \mid x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j = 0, \quad i = 1, \dots, k, \quad x_i \geq 0, \quad i = 1, \dots, n\}.$$

*If  $\bar{x}$  is an extreme ray of*

$$\begin{aligned} x_{B_i^t} + \sum_{j \in N^t} a_{ij}^t x_j &= 0, \quad i = 1, \dots, l \leq k, \\ x_i &\geq 0, \quad i = 1, \dots, n. \end{aligned}$$

such that  $\sum_{j \in N^t} w_j^t \bar{x}_j \geq 0$ , then there does not exist an extreme ray  $\hat{x}$  of  $C$  such that  $\sum_{j \in N^t} w_j^t \hat{x}_j < 0$ .

**Proposition 5.20** *If the simplex algorithm with the Cacioppi and Schrage entering and exit rules is applied to a degenerate tableau (5.36)-(5.37), then after a finite number of simplex pivots either an extreme ray with negative cost is found, or there are no level  $\infty$  variables with a negative reduced cost which implies that there is no extreme ray of the cone with a negative cost.*

**Proof:** Proof by induction on  $k$ . Consider  $k = 1$ . All nonbasic variables are initially assigned level  $\infty$ . Assume in tableau  $t$ , variable  $h$  with level  $\infty$  has a negative reduced cost. Then  $a_{1h}^t \leq 0$  or  $a_{1h}^t > 0$ . If  $a_{1h}^t \leq 0$  then we have an extreme ray of the cone and are done. If  $a_{1h}^t > 0$  then basic variable  $x_{B_1^t}$  exists in the basis and is assigned a level of 0. A level 0 variable is never again eligible to enter the basis. After at most  $|N^t|$  pivots an extreme ray with negative cost is found or all variables get assigned a level of 0 and are no longer eligible to enter the basis. In this case, by Lemma 5.18 there is no extreme ray of cone with negative cost.

Now assume the result is true for a cone with  $k$  constraints in (5.36) and show this implies the result for  $k + 1$  constraints. If the Cacioppi and Schrage Rule is applied to (5.36)-(5.37) with  $k + 1$  constraints in (5.36) there are three possible outcomes. The first outcome is that simplex terminates because all level  $\infty$  variables have a nonnegative reduced cost. In this case by Lemma 5.18 we reach the correct conclusion that there is no extreme ray of the degeneracy cone with a negative cost.

The second outcome is that an extreme ray of the degeneracy cone is found with negative cost. In this case we terminate with the desired extreme ray.

The third outcome is that simplex cycles. This cannot happen. Whenever a variable  $x_{B_1^r}$  exits the basis at pivot  $r$  it is assigned level 0 and all other nonbasic variables are assigned level  $\infty$ . Once a nonbasic variable is assigned level 0 in a problem it can never enter the basis again since it will never be reassigned a level greater than 0. This has the effect of eliminating variable  $x_{B_1^r}$  from the variable list and starting the problem over from scratch with one less variable. This is valid by Lemma 5.15. Since there are finite number of nonbasic variables, a basic variable in row 1 can exit the basis a finite number of times. Let pivot  $r$  be the last pivot on a variable in row 1. At the start of pivot  $r + 1$  and nonbasic variables have level  $\infty$  and there are no more pivots in row 1. The problem is then reduced to a  $k$  row problem and by the induction hypothesis

simplex must terminate after a finite number of pivots with an extreme ray of the cone with negative cost involving the last  $k$  rows of the cone, or correctly conclude that none exist. In the latter case of no extreme ray with negative cost, by Lemma 5.19 there is no extreme ray for the  $k+1$  constraint cone with negative cost. If a negative cost extreme ray is found at pivot  $t > r$  for entering variable  $l$ , we have either  $a_{1l}^t \leq 0$  and there is an extreme ray of the entire cone, or  $a_{1l}^t > 0$  which results in a pivot on row 1. But we assumed we were done with such pivots so the proposition is proved.  $\square$

Anti-cycling rules for the simplex algorithm have been the subject of considerable study. The earliest anti-cycling rules are due to Charnes [88] and Dantzig, Orden, and Wolfe [111]. The method of Dantzig, Orden, and Wolfe is based on a *lexicographic* ordering of the columns of the updated tableau. The method of Charnes is based on perturbing the right hand side vector and can be shown to be equivalent to the method of Dantzig, Orden and Wolfe. Bland [62] proved finite convergence of simplex using a very simple entering and exit variable rule. Select the entering variable as the variable with the smallest index and a negative reduced cost. Select the exit variable using the minimum ratio test and break ties by selecting the variable with the minimum index. More recently, Zhang [474] has developed a general rule to guarantee finite convergence which subsumes the Bland's rule.

There are several characteristics to look for in selecting a rule to prevent cycling. First, it should be easy to implement. Bland's rule is certainly the easiest rule to implement. Second, there should be as much flexibility as possible. Bland's rule is also the least flexible. The Cacioppi and Schrage rule is particularly flexible in the selection of the entering variable since any variable with a level of  $\infty$  is eligible. It also has the advantage that variables are eliminated from consideration, it is not necessary to price out any variable with a level less than  $\infty$ .

All of the anti-cycling rules cited guarantee that the simplex algorithm will converge in a finite number of pivots. However, these rules do not guarantee that the number of consecutive degenerate pivots will be a polynomial function of the number of variables  $n$  in the problem and the number of rows  $m$ . If fact, Megiddo [323] has shown that if such an anti-cycling rule existed then there is an algorithm for solving the linear programming problem which is a polynomial function of the number of variables  $n$  and the number of rows  $m$ , i.e. a *strongly polynomial* algorithm for solving linear programs. When the number of consecutive degenerate pivots is an exponential function of  $n$  and  $m$  then this phenomenon is called *stalling*. In Chapter 14 we study network flow problems. For these problems it is possible to give anti-stalling rules.

**Example 5.21 (Example 5.14 Continued.)** In Example 5.14,  $k = 2$ . In the third tableau variable  $x_2$  is pivoted out of the tableau and by Lemma 5.16 variable  $x_2$  can be removed from the candidate list of entering variables. Now consider tableau six. Variable  $x_2$  is selected as the entering variable based on its reduced cost. However, it is no longer a candidate variable and the cycle is broken.

## 5.7 COMPLEXITY OF THE SIMPLEX ALGORITHM

Given appropriate anti-cycling strategies, the simplex algorithm is finite. However, it is not a polynomial algorithm. Each pivot of the simplex algorithm requires updating the basis inverse and performing an FTRAN and BTRAN. It is not hard to show that the total number of arithmetic operations for each pivot is  $O(mn)$ . In practice, the simplex algorithm is always implemented on a finite precision machine. However, using the methods described in Appendix B, each simplex pivot can be done in polynomial time in infinite precision. Therefore, proving that the simplex algorithm has a running time which is a polynomial function of the number of variables, constraints and binary encoding of the data requires providing a polynomial bound on the number of pivots. No one has done this. However, examples exist showing that an exponential number of pivots are required for given pivot rules. Klee and Minty [266] were the first to provide an example where the simplex algorithm using the Dantzig rule of selecting the entering variable with the largest reduced cost requires  $O(2^n)$  pivots. This does not imply one could not construct a pivot rule for which only a polynomial number of pivots is required. Most experts in the area probably feel that no such rule exists.

Projection was the first algorithm for solving linear programs. In the worst case it is a doubly exponential algorithm. The simplex method is not a polynomial algorithm. The first polynomial time algorithm for linear programming was finally discovered in 1979 by L.G. Khachiyan. See [263] and [262]. This algorithm is often referred to as the *ellipsoid algorithm*. Although it has extremely important implications in combinatorial optimization, see for example Grötschel, Lovász, and [212], it is not practical from a computational standpoint. In the next two chapters we develop polynomial algorithms for linear programming which have met with spectacular success in solving large, sparse problems.

## 5.8 CONCLUSION

The revised simplex algorithm is probably the most important algorithm ever developed in the field of linear optimization. Although the interior point methods described in the next two chapters may be faster, it is the simplex algorithm that made the field of operations research what it is today. One of the most significant developments in the simplex algorithm was the revised simplex algorithm using the product form of the inverse . This was due to Dantzig and Orchard-Hays [110]. It is an excellent example of the crucial role played by numerical linear algebra in optimization. The role of numerical linear algebra in linear programming is the central theme of Chapter 13.

Finally, we highly recommend the Dantzig book [108]. It is the classic text in the area. The book by Orchard-Hays [358] is also a classic and was the first to emphasize implementing the revised simplex algorithm in a modern (for the time) computing environment. A very cogent description of a modern simplex implementation is given in Bixby [55]. The bibliography in the Bixby article lists many of the classic papers in the development of linear programming.

## 5.9 EXERCISES

- 5.1 If the linear program  $\min\{c^T x \mid Ax = b, x \geq 0\}$  is unbounded, the simplex algorithm will detect this fact. Show how to construct an extreme ray  $\bar{x}$  of the recession cone  $\{x \mid Ax = 0, x \geq 0\}$  such that  $c^T \bar{x} < 0$  from the simplex tableau at the pivot where simplex discovers that the problem is unbounded.
- 5.2 Apply the simplex algorithm to Example 5.2 using an objective function of  $2x_1 - 8x_2$  and starting point  $(0, 6)$ . When simplex discovers that this problem is unbounded use your method of question 1 to construct the appropriate extreme ray.
- 5.3 Assume that at pivot  $k$  of the simplex algorithm variable  $x_i$  leaves the basis. Is it possible for variable  $x_i$  to enter the basis again at pivot  $k+1$ ? Either give an example of this behavior or prove that it cannot happen.
- 5.4 Assume that at pivot  $k$  of the simplex algorithm variable  $x_i$  enters the basis. Is it possible for variable  $x_i$  to leave the basis at pivot  $k+1$ ? Either give an example of this behavior or prove that it cannot happen.

### 5.5 Take the linear program<sup>2</sup>

$$\begin{aligned}
 \max \quad & 10x_1 + 9x_2 \\
 (7/10)x_1 + x_2 &\leq 630 \\
 (1/2)x_1 + (5/6)x_2 &\leq 600 \\
 x_1 + (2/3)x_2 &\leq 708 \\
 (1/10)x_1 + (1/4)x_2 &\leq 135 \\
 x_1, x_2 &\geq 0
 \end{aligned}$$

and solve this linear program with the simplex algorithm using full tableaus as in Section 5.3. Use the origin as the starting solution.

### 5.6 Take the linear program

$$\begin{aligned}
 \max \quad & 10x_1 + 9x_2 \\
 (7/10)x_1 + x_2 &\leq 630 \\
 (1/2)x_1 + (5/6)x_2 &\leq 600 \\
 x_1 + (2/3)x_2 &\leq 708 \\
 (1/10)x_1 + (1/4)x_2 &\leq 135 \\
 x_1, x_2 &\geq 0
 \end{aligned}$$

and solve this linear program with revised simplex using an explicit form of the basis inverse. Use the origin as the starting solution.

### 5.7 Take the linear program

$$\begin{aligned}
 \max \quad & 10x_1 + 9x_2 \\
 (7/10)x_1 + x_2 &\leq 630 \\
 (1/2)x_1 + (5/6)x_2 &\leq 600 \\
 x_1 + (2/3)x_2 &\leq 708 \\
 (1/10)x_1 + (1/4)x_2 &\leq 135 \\
 x_1, x_2 &\geq 0
 \end{aligned}$$

and solve this linear program with revised simplex using the product form of the basis inverse. Use the origin as the starting solution.

### 5.8 Consider the linear program $\max\{c^T x \mid Ax = b\}$ . Show that this linear program either does not have an optimal solution, or every feasible solution is optimal.

---

<sup>2</sup>This is the Par Inc. problem from Anderson, Sweeney and Williams [12].

5.9 Prove Lemma 5.15.

5.10 In many applications a decision variable  $x$  is unrestricted in sign. For example,  $x$  might represent the change in inventory level between two time periods. This could either be positive or negative. One method for converting a problem with unrestricted variables into standard form is to replace every appearance of  $x$  in the model formulation with  $x_1 - x_2$  and require  $x_1, x_2$  to be nonnegative. Prove that in basic feasible solution it will never be the case that both  $x_1$  and  $x_2$  are strictly positive.

5.11 Convert the following

$$\begin{aligned} & \min x_1 + x_2 + 5|x_3| \\ \text{s.t. } & x_1 + x_2 + 7x_3 \leq 5 \\ & x_1 - 4x_2 \geq 11 \\ & x_1 \geq 0 \end{aligned}$$

into a linear program in standard form.

5.12 Assume  $A\bar{x} = b$ ,  $\bar{x} \geq 0$  but  $\bar{x}$  is not necessarily basic, i.e. it may strictly more than  $m$  positive components. Describe a pivoting algorithm which will start with  $\bar{x}$  and move to a basic feasible solution  $\hat{x}$  such that  $c^T\hat{x} \leq c^T\bar{x}$ . You may assume that the linear program has an optimal solution.

5.13 Assume that a basis matrix with  $d$  nonzero elements can be put into either upper or lower triangular form by appropriate ordering of the variables and row. How many nonzero elements will the eta vectors contain in the product form of the inverse?

5.14 Take the first tableau of Example 5.14 and apply the Cacioppi and Schrage rule. Indicate the level of each nonbasic variable at each pivot.

5.15 Prove Lemma 5.19.

5.16 Prove that if there is a polynomial algorithm to answer the decision problem "is the polyhedron  $\{x \in \Re^n \mid Ax = b, x \geq 0\}$ , nonempty," there is a polynomial algorithm to solve the optimization problem  $\min\{c^T x \mid Ax = b, x \geq 0\}$ .

5.17 Prove that if  $M$  is a nonsingular  $m \times m$  matrix,  $u, v$  are  $m$  vectors and  $1 + v^T M^{-1} u \neq 0$ , then

$$(M + uv^T)^{-1} = M^{-1} - \frac{M^{-1}uv^TM^{-1}}{1 + v^TM^{-1}u}.$$

- 5.18 Prove Proposition 5.7.
- 5.19 Explain why Table 5.2 probably overstates the work required for the revised simplex with the product form of the inverse.

---

## MORE ON SIMPLEX

### 6.1 INTRODUCTION

In the previous chapter, the basics of the simplex and revised simplex algorithms were developed. Degeneracy and convergence issues were addressed. In this chapter we continue with several enhancements to the basic algorithm and related issues. In Section 6.2 we continue the development of sensitivity analysis which was first discussed in Chapters 2 and 3. We show that an analysis of the simplex tableau provides dual variable values, reduced costs and range analysis for the right hand side and objective function, but the simplex algorithm does not provide results which are as reliable as those obtained using projection and inverse projection. In Section 6.3 we develop the dual simplex algorithm. This is the simplex algorithm applied to the dual problem but using the primal simplex tableau. In our earlier development of the simplex algorithm we assumed that all variables had a lower bound of zero and an upper bound of  $\infty$ . In Section 6.4 we show how to incorporate variables with arbitrary lower and upper bounds into the simplex algorithm. The simplex algorithm requires an initial feasible basis. In Section 6.5 we show how to find an initial basic feasible solution. In Step 3 of the simplex algorithm it is necessary to select a variable to enter the basis. In Section 6.6 we describe several methods for selecting the entering variable. In Section 6.7 we discuss other numerical issues such as tolerances, the minimum ratio test and scaling. Concluding remarks are given in Section 6.8. Exercises are provided in Section 6.9. One topic not covered in this chapter is preprocessing. By preprocessing we mean analyzing the problem prior to solution in order to discover obvious redundancies, tighter variable bounds, eliminate variables and constraints, etc. This is very important in reducing the solution time for solving linear programs. Our discussion of preprocessing linear programs is postponed until Chapter 15.

## 6.2 SENSITIVITY ANALYSIS

Sensitivity analysis was introduced in the Chapter 2. The sensitivity analysis was based on projection and inverse projection. We now perform sensitivity analysis of the right hand side vector and the objective function coefficient vector based on the optimal linear simplex tableau.

### 6.2.1 Right Hand Side Analysis

Estimating how the optimal objective function value of a linear program changes as a function of the right hand side is an important problem. For example, a linear program to calculate optimal production schedules might be solved on a monthly basis. This type of model will have demand constraints which say that demand must be met from inventory or production. The product demand is the right hand side in these constraints. It is important to accurately estimate changes in cost or profit as this demand vector changes from month to month. Also, if there are constraints on scarce resources, the dual variables provide a price on the scarce resources at the margin and right hand side sensitivity analysis provides information on the validity of the dual variable value as the right hand side changes. In Chapter 2 the right hand side sensitivity analysis was based on the solution produced using projection. In this section we study the same topic within the context of the simplex algorithm.

Consider a linear program in standard form.

$$(LP) \quad \begin{aligned} & \min c^\top x \\ & \text{s.t. } Ax = b \\ & \quad x \geq 0 \end{aligned}$$

If  $(\bar{x}_B, \bar{x}_N)$  is an optimal basic feasible solution to this linear program with basis  $B$ , and each right hand side  $b_i$  is altered by the quantity  $\delta_i$ , which may be positive or negative, then the new tableau is

$$\begin{aligned} z_0 - \sum_{j \in N} \bar{w}_j x_j &= c_B^\top \bar{x}_B \\ x_{B_i} + \sum_{j \in N} \bar{a}_{ij} x_j &= (\bar{b}_i + \bar{\delta}_i), \quad i = 1, \dots, m, \\ x_i &\geq 0, \quad i = 1, \dots, n \end{aligned}$$

where  $\bar{\delta} = A_B^{-1}\delta$ . The current basis remains optimal if and only if  $(\bar{b}_i + \bar{\delta}_i) \geq 0$  for  $i = 1, \dots, m$ . This is easy to see because  $(\bar{b}_i + \bar{\delta}_i) \geq 0$  implies that the

basis remains feasible and  $\tilde{\delta}_i$  does not affect the reduced costs  $\bar{w}_j$ . Therefore, dual feasibility and complementary slackness are not affected by  $\tilde{\delta}_i$ . In matrix notation,  $A = [A_B, A_N]$ ,  $A_B x_B + A_N x_N = b + \delta$  and multiplying this equation by  $A_B^{-1}$  gives

$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b + A_B^{-1} \delta = \bar{b} + \bar{\delta}.$$

Each  $\bar{\delta}_i$  represents the inner product of row  $i$  of  $A_B^{-1}$  with  $\delta$ . Denote element  $(i, j)$  of  $A_B^{-1}$  by  $\alpha_{ij}$ . Then the basis associated with  $B$  is optimal if and only if

$$\bar{b}_i + \sum_{j=1}^m \alpha_{ij} \delta_j \geq 0, \quad i = 1, \dots, m.$$

Clearly analyzing multiple changes in the right hand side is complicated. However, if only one right hand side is allowed to change the analysis simplifies considerably. Assume  $\delta_i = 0$  for all  $i \neq k$ . Then the solution associated with basis  $B$  remains optimal if and only if

$$\bar{b}_i + \alpha_{ik} \delta_k \geq 0, \quad i = 1, \dots, m. \quad (6.1)$$

The simplex algorithm determines the *allowable increase* on the right hand side of constraint  $k$  to be the largest positive value of  $\delta_k$  such that (6.1) is satisfied, that is the allowable increase on the right hand side of constraint  $k$  is

$$\Delta_k^+ := \min\{-\bar{b}_i / \alpha_{ik} \mid \alpha_{ik} < 0, i = 1, \dots, m\}. \quad (6.2)$$

If there are no negative elements in column  $k$  of  $A_B^{-1}$  define  $\Delta_k^+ := \infty$ . In words,  $\Delta_k^+$  is the maximum amount that the right hand side of constraint  $k$  can increase without making the current basis infeasible. Similarly, when the right hand side is decreased, only the positive  $\alpha_{ik}$  are considered and the allowable decrease on the right hand side of constraint  $k$  is

$$\Delta_k^- := \min\{\bar{b}_i / \alpha_{ik} \mid \alpha_{ik} > 0, i = 1, \dots, m\} \quad (6.3)$$

If there are no positive elements in column  $k$  of  $A_B^{-1}$ , define  $\Delta_k^- := \infty$ . From equation (6.1) and the definition in (6.2) and (6.3) increasing (decreasing) the right hand side of constraint  $k$  by the allowable increase (decrease) results in a degenerate solution. In Chapter 3 we defined the allowable increase (decrease) on the right hand side of constraint  $k$  with respect to a specific dual solution  $\bar{u}$ . It was defined as the maximum increase (decrease) in a right hand side such that the  $\bar{u}$  remained dual optimal. These two definitions are related by the following Proposition which is left as an exercise.

**Proposition 6.1** *If  $(\bar{x}_B, \bar{x}_N)$  is an optimal basic feasible solution, and the dual solution  $(\bar{u}, \bar{w})$  where  $\bar{u}^\top = c_B^\top A_B^{-1}$ ,  $\bar{w}_B = 0$ , and  $\bar{w}_N^\top = c_N^\top - \bar{u}^\top A$  is not*

degenerate, then the allowable increase on the right side of constraint  $k$  such that  $\bar{u}^\top = c_B^\top A_B^{-1}$  remains dual optimal is given by (6.2). Similarly for the allowable decrease.

The allowable increase and decrease given in (6.2)-(6.3) by the simplex method is conservative. If  $(\bar{x}, \bar{u})$  are a primal-dual optimal pair and if  $b_k$  is changed by its allowable increase (decrease)  $\bar{u}$  remains dual optimal. Of course it may be possible to change  $b_k$  by more than the values given in (6.2)-(6.3) and have  $\bar{u}$  remain dual optimal.

**Example 6.2 (Example 2.42)** This example is put into standard form by adding a nonnegative slack variable to constraints (E1) - (E4). Recall that the true allowable decrease on constraint (E3) is 3.0. However, when a simplex code finds the optimal primal solution of  $x_1 = 2.0$  and  $x_2 = 2.0$  with positive slack on constraints (E1) and (E2), it will report the allowable decrease on constraint (E3) as 2.0. The slack variable for constraint (E4) is not in the basis. Then if (E3) is changed to  $-x_2 \geq -4 - \epsilon$  for  $\epsilon > 0$ , the optimal primal solution must have positive slack on constraint (E4) so there is a basis change. However, the optimal dual solution does not change.

Obviously, we would like the “true” allowable increase and decrease. Unfortunately, this requires using projection (see Section 2.8) or more simplex pivoting which are considerably much more work than the calculations in (6.2)-(6.3). Many linear programming codes such as LINDO, will print a *parametric analysis* of a right hand side coefficient at the request of the user. The LINDO parametric analysis for constraint (E3) for a right hand side range of -2 to -10 is given below.

	VAR OUT	VAR IN	PIVOT ROW	RHS VAL	DUAL PRICE BEFORE PIVOT	OBJ VAL
				-2.00000	-1.00000	-2.00000
SLK	3	SLK	5	3 -4.00000	+1.00000	-4.00000
SLK	2	SLK	4	2 -5.00000	-1.00000	-5.00000
				-10.0000	0.000000E+00	-5.00000

Notice that the dual variable remains at 1 (the dual price is the negative of the dual variable value) as the right hand side decreases from -4, all the way to -5. Also, the dual variable becomes *smaller* as the right hand side becomes smaller. This is an illustration of Lemma 2.40 and the more general result, Corollary

2.49 which states that the optimal value function of the right hand side of a minimization problem is a piecewise linear convex function. The endpoints provided by a range analysis from simplex will lie inside the breakpoints of the function. Finally, the reader is cautioned on using the dual variable of a constraint whenever the allowable increase or decrease of the right hand side of the constraint is zero. If, for example, the right hand side allowable increase is zero, this means that the dual variable may change in value if the right hand side is increased by any positive amount.

Equations (6.2)-(6.3) define the allowable increase and decrease on the right hand side of a constraint  $k$  under the assumption that  $b_k$  is the only right hand side that changes. If several right hand sides change simultaneously the following proposition, sometimes called *100% Rule* provides a sufficient condition for maintaining primal feasibility.

**Proposition 6.3 (Right Hand Side 100% Rule)** *If  $\delta_k$  is the change in the right hand side vector  $b_k$  for  $k = 1, \dots, m$  and*

$$\sum_{\substack{k=1 \\ \delta_k > 0}}^m |\delta_k|/\Delta_k^+ + \sum_{\substack{k=1 \\ \delta_k < 0}}^m |\delta_k|/\Delta_k^- \leq 1.0$$

*then the current basis remains optimal.*

### Proof:

The definition of  $\Delta_k^+$  and  $\Delta_k^-$  and equation (6.1) imply for a given  $i^*$

$$\bar{b}_{i^*} + \alpha_{i^*k}\Delta_k^+ \geq 0 \quad \text{for all } \alpha_{i^*k} < 0, \quad k = 1, \dots, m, \quad (6.4)$$

$$\bar{b}_{i^*} - \alpha_{i^*k}\Delta_k^- \geq 0 \quad \text{for all } \alpha_{i^*k} > 0, \quad k = 1, \dots, m. \quad (6.5)$$

For the given  $i^*$ , in (6.4) multiply by  $|\delta_k|/\Delta_k^+$ , for  $k = 1, \dots, m$  with  $\alpha_{i^*k} < 0$  and in (6.5) multiply by  $|\delta_k|/\Delta_k^-$  for all  $k = 1, \dots, m$  with  $\alpha_{i^*k} > 0$ . Adding the result and assuming the worst case scenario which is  $\delta_k > 0$  if  $\alpha_{i^*k} < 0$  and  $\delta_k < 0$  if  $\alpha_{i^*k} > 0$  gives

$$\begin{aligned} \bar{b}_{i^*} + \sum_{k=1}^m \alpha_{i^*k}\delta_k &\geq \sum_{\substack{k=1 \\ \alpha_{i^*k} < 0}}^m (|\delta_k|/\Delta_k^+) \bar{b}_{i^*} + \sum_{\substack{k=1 \\ \alpha_{i^*k} < 0}}^m \alpha_{ik}|\delta_k| \\ &\quad + \sum_{\substack{k=1 \\ \alpha_{i^*k} > 0}}^m (|\delta_k|/\Delta_k^-) \bar{b}_{i^*} - \sum_{\substack{k=1 \\ \alpha_{i^*k} > 0}}^m \alpha_{ik}|\delta_k| \geq 0, \end{aligned}$$

where the first inequality follows directly from the hypothesis and the second inequality follows from (6.4) - (6.5).  $\square$

**Example 6.4** The inverse for the optimal basis and the updated right hand side from Example 5.2 are

$$A_B^{-1} = \begin{bmatrix} -1 & 0 & 2 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 2 & -2 \\ 0 & -1 & 4 & -3 \end{bmatrix}, \quad A_B^{-1}b = \begin{bmatrix} 9 \\ 10 \\ 4 \\ 12 \end{bmatrix}$$

Calculate the allowable increase of each constraint right hand side using (6.2).

$$\begin{aligned}\Delta_1^+ &= \min\{-9/-1\} = 9 \\ \Delta_2^+ &= \min\{-12/-1\} = 12 \\ \Delta_3^+ &= \infty \\ \Delta_4^+ &= \min\{-9/-1, -10/-1, -4/-2, -12/-3\} = 2\end{aligned}$$

Calculate the allowable decrease of each constraint right hand side using (6.3).

$$\begin{aligned}\Delta_1^- &= \infty \\ \Delta_2^- &= \infty \\ \Delta_3^- &= \min\{9/2, 10/2, 4/2, 12/3\} = 2 \\ \Delta_4^- &= \infty\end{aligned}$$

## 6.2.2 Objective Function Coefficient Analysis

It is also important to understand how the optimal objective function value changes with changes in the objective function coefficients. For example, perhaps a product is not in the optimal manufacturing mix. How much would the price of the product have to increase before the product became part of the optimal manufacturing mix? If a product is part of an optimal mix, but the price must be reduced by 10%, is it still optimal to produce this product? Such questions are answered by objective function coefficient analysis.

Again, assume a linear program in standard form and that  $(\bar{x}_B, \bar{x}_N)$  is an optimal basic feasible solution to this linear program with basis  $B$ , and each objective function coefficient is altered by the quantity  $\delta_j$ , which may be positive or negative. The updated tableau is

$$z_0 - \sum_{j \in N} (\bar{w}_j + \bar{\delta}_j)x_j = c_B^\top \bar{x}_B,$$

$$\begin{aligned} x_{B_i} + \sum_{j \in N} \bar{a}_{ij} x_j &= \bar{b}_i, \quad i = 1, \dots, m, \\ x_i &\geq 0, \quad i = 1, \dots, n \end{aligned}$$

where  $\bar{\delta}_N^\top = \delta_N^\top - \delta_B^\top A_B^{-1} A_N$ . The current basis remains optimal if and only if  $(\bar{w}_j + \bar{\delta}_j) \geq 0$  for  $j \in N$ . This is because  $(\bar{w}_j + \bar{\delta}_j) \geq 0$  implies that the dual solution  $\bar{u}^\top = (c_B + \delta_B)^\top A_B^{-1}$  is still dual feasible, the primal solution is unaffected by the  $\delta_j$  on objective function coefficients and remains feasible, and complementary slackness is unaffected.

Each  $\bar{\delta}_j$  for  $j \in N$  is the change in the objective coefficient  $c_j$  minus the inner product of the vector of changes in the objective function coefficients of the basic variables with the updated column  $j$ . Then the current basis remains optimal if and only if

$$\bar{w}_j + \delta_j - \sum_{i=1}^m \delta_{B_i} \bar{a}_{ij} \geq 0, \quad j \in N. \quad (6.6)$$

Again, to simplify the analysis assume that only one objective function coefficient changes and we are interested in the maximum possible change so that the current optimal basis remains optimal. If the coefficient that changes is a nonbasic variable coefficient then the allowable increase and decrease are

$$\Theta_j^+ := \infty, \quad (6.7)$$

$$\Theta_j^- := \bar{w}_j. \quad (6.8)$$

If the coefficient that changes is the coefficient of the basic variable  $x_{B_k}$ , then (6.6) implies

$$\bar{w}_j - \delta_{B_k} \bar{a}_{kj} \geq 0, \quad j \in N, \quad (6.9)$$

and the allowable increase and decrease are

$$\Theta_{B_k}^+ := \min\{\bar{w}_j / \bar{a}_{kj} \mid \bar{a}_{kj} > 0, j \in N\}, \quad (6.10)$$

$$\Theta_{B_k}^- := \min\{-\bar{w}_j / \bar{a}_{kj} \mid \bar{a}_{kj} < 0, j \in N\}. \quad (6.11)$$

If  $\bar{a}_{kj} \leq 0, j \in N$  then  $\Theta_{B_k}^+ := \infty$ . If  $\bar{a}_{kj} \geq 0, j \in N$  then  $\Theta_{B_k}^- := \infty$ .

In Chapter 3 we defined the allowable increase (decrease) on an objective function coefficient with respect to a specific optimal primal solution  $\bar{x}$ . It was defined as the maximum increase (decrease) in a given objective function coefficient such that  $\bar{x}$  remained primal optimal. This definition is related to the simplex based definition in the following proposition which is "dual" to Proposition 6.1

**Proposition 6.5** If  $(\bar{x}_B, \bar{x}_N)$  is an optimal primal basic feasible solution which is not degenerate, then the maximum allowable increase on the objective function coefficient of variable  $j$  such that  $(\bar{x}_B, \bar{x}_N)$  remains primal optimal is given by (6.10). The allowable decrease is defined in a similar fashion.

By Proposition 6.5 the allowable increase and decrease calculated using (6.10)-(6.11) and (6.7)-(6.8) is conservative. If an objective function coefficient is changed by these amounts the current basis remains optimal. However, it may be possible to change the objective function coefficients by more than the amount given by these formulas and have the optimal primal basis remain optimal. Changing an objective function coefficient by more than the amounts given in (6.10)-(6.11) and (6.7)-(6.8) will result in a negative reduced cost. However, if there is primal degeneracy then it may be possible to pivot on the column with the negative reduced cost and not change the values of the primal variables. See also Proposition 3.10 in Chapter 3.

There is also an analog to the 100% rule for right hand side sensitivity analysis.

**Proposition 6.6 (Objective Function 100% Rule)** If  $\delta_{B_k}$  is the change in the objective function coefficient  $c_{B_k}$  for  $k = 1, \dots, m$  and

$$\sum_{\substack{j=1 \\ \delta_{B_k} > 0}}^m |\delta_{B_k}| / \Theta_{B_k}^+ + \sum_{\substack{j=1 \\ \delta_{B_k} < 0}}^m |\delta_{B_k}| / \Theta_{B_k}^- \leq 1.0$$

then the current basis remains optimal.

**Example 6.7** The optimal tableau from Example 5.2 is

$$\begin{array}{rccccccccc} z_0 & & -2x_5 & -x_6 & = & -22 \\ x_3 & & +2x_5 & -x_6 & = & 9 \\ x_2 & & +2x_5 & -x_6 & = & 10 \\ x_1 & & +2x_5 & -2x_6 & = & 4 \\ x_4 & & +4x_5 & -3x_6 & = & 12 \end{array}$$

Calculate the allowable increase using ((6.7)) and (6.10 ).

$$\begin{aligned} \Theta_3^+ &= \min\{2/2\} = 1 \\ \Theta_2^+ &= \min\{2/2\} = 1 \end{aligned}$$

$$\begin{aligned}\Theta_1^+ &= \min\{2/2\} = 1 \\ \Theta_4^+ &= \min\{2/4\} = 1/2 \\ \Theta_5^+ &= \infty \\ \Theta_6^+ &= \infty\end{aligned}$$

Calculate the allowable decrease using (6.8) and (6.11).

$$\begin{aligned}\Theta_3^- &= \min\{-1/-1\} = 1 \\ \Theta_2^- &= \min\{-1/-1\} = 1 \\ \Theta_1^- &= \min\{-1/-2\} = 1/2 \\ \Theta_4^- &= \min\{-1/-3\} = 1/3 \\ \Theta_5^- &= 2 \\ \Theta_6^- &= 1\end{aligned}$$

### 6.3 THE DUAL SIMPLEX ALGORITHM

The primal simplex algorithm starts with a primal basic feasible solution and complementary dual solution, which is not necessarily feasible, and iterates to dual feasibility. The dual simplex algorithm due to Lemke [283] starts with a dual basic feasible solution and complementary primal solution, which is not necessarily feasible, and iterates to primal feasibility. The dual of linear program (*LP*) is

$$(DLP) \quad \begin{aligned} &\max b^\top u \\ \text{s.t.} \quad &A_B^\top u + w_B = c_B \\ &A_N^\top u + w_N = c_N \\ &w_B, w_N \geq 0.\end{aligned}$$

By the strong duality theorem, if there is an optimal solution to the primal linear program (*LP*), there is an optimal solution to the dual linear program (*DLP*). By the fundamental theorem of linear programming, if there is an optimal solution to the primal linear program (*LP*), then there is an optimal basic feasible solution. Assume (*LP*) has an optimal solution and  $\hat{B}$  indexes an optimal basis. Then an optimal primal solution is  $x_{\hat{B}} = A_{\hat{B}}^{-1}b$  and the corresponding feasible dual solution is

$$\hat{u}^\top = c_{\hat{B}}^\top A_{\hat{B}}^{-1} \quad w_{\hat{N}}^\top = c_N^\top - \hat{u}^\top A_N, \quad w_{\hat{B}} = 0. \quad (6.12)$$

It follows from complementary slackness that the dual solution is also optimal. Then for every optimal primal basis matrix  $\hat{B}$ , there is a corresponding optimal

dual basic feasible solution given by (6.12) and corresponding optimal dual basis matrix.

$$\begin{bmatrix} A_{\hat{B}}^\top & 0 \\ A_N^\top & I \end{bmatrix} \quad (6.13)$$

By definition of basis,  $A_{\hat{B}}$  has rank  $m$  and the matrix in (6.13) has full rank, which is  $n$ , so the dual solution given in (6.12) is a dual basic feasible solution. Thus, if the simplex algorithm is applied to the dual problem we only need to consider dual basic feasible solutions with basis (6.13) derived from the corresponding primal basis. The inverse of the dual basis matrix (now using  $B$  instead of  $\hat{B}$  to denote a generic basis, not necessarily the optimal one) is

$$\begin{bmatrix} A_B^{-\top} & 0 \\ -A_N^\top A_B^{-\top} & I \end{bmatrix}. \quad (6.14)$$

In (6.14),  $A_B^{-\top}$  is used to denote the inverse matrix of  $A_B^\top$ . In examining the logic behind dual simplex, we follow the development of primal simplex and look at the projection of the dual problem into the space of the dual nonbasic variables. Given a basis  $B$  rewrite the dual problem as

$$(DLP) \quad \begin{aligned} & \max b^\top u \\ \text{s.t.} \quad & \begin{bmatrix} A_B^\top \\ A_N^\top \end{bmatrix} u + \begin{bmatrix} w_B \\ w_N \end{bmatrix} = \begin{bmatrix} c_B \\ c_N \end{bmatrix} \\ & w_B, w_N \geq 0. \end{aligned}$$

Multiply the dual constraints by the inverse of the dual basis matrix given in (6.14).

$$(DLP) \quad \begin{aligned} & \max b^\top u \\ \text{s.t.} \quad & \begin{bmatrix} I \\ 0 \end{bmatrix} u + \begin{bmatrix} A_B^{-\top} w_B \\ -A_N^\top A_B^{-\top} w_B + w_N \end{bmatrix} = \begin{bmatrix} A_B^{-\top} c_B \\ -A_N^\top A_B^{-\top} c_B + c_N \end{bmatrix} \\ & w_B, w_N \geq 0. \end{aligned}$$

Substitute out the basic dual variables using

$$u = A_B^{-\top} c_B - A_B^{-\top} w_B \quad (6.15)$$

$$w_N = (-A_N^\top A_B^{-\top} c_B + c_N) + A_N^\top A_B^{-\top} w_B. \quad (6.16)$$

The dual linear program in the projected space of the nonbasic dual variables  $w_B$  is

$$\max b^\top A_B^{-\top} c_B - b^\top A_B^{-\top} w_B$$

$$(DLP(w_B)) \quad \text{s.t.} \quad -A_N^T A_B^{-\top} w_B \leq c_N - A_N^T A_B^{-\top} c_B \\ w_B \geq 0.$$

The objective function is equivalent to

$$-b^T A_B^{-\top} c_B + \min b^T A_B^{-\top} w_B.$$

Thus, the *dual reduced costs* are  $b^T A_B^{-\top}$ . But these are the primal solution values associated with the primal basis matrix  $A_B$ . That is,  $\bar{x}_B = A_B^{-1}b$ . The structure of  $(DLP(w_B))$  implies the following Lemma.

**Lemma 6.8** *If  $B$  is basis such that  $\bar{x}_B = A_B^{-1}b \geq 0$  and  $c_N - A_N^T A_B^{-\top} c_B \geq 0$ , then  $\bar{w}_B = 0$  is an optimal solution to  $(DLP(w_B))$ .*

**Proof:** If  $c_N - A_N^T A_B^{-\top} c_B \geq 0$  then  $\bar{w}_B = 0$  is a feasible solution to  $(DLP(w_B))$ . If  $\bar{x}_B = A_B^{-1}b \geq 0$ , then the objective function coefficients for the minimization version of the objective function are nonnegative. Then  $\bar{w}_B = 0$  is also optimal.  $\square$

What if  $c_N - A_N^T A_B^{-\top} c_B \geq 0$  but  $\bar{x}_B = A_B^{-1}b$  is not nonnegative? If  $\bar{x}_{B_{i^*}} < 0$ , then the feasible solution  $\bar{w}_B = 0$  is improved by increasing  $w_{B_{i^*}}$ . How much can this nonbasic dual variable increase? If  $w_{B_i} = 0$  for  $i \neq i^*$ , then  $w_{B_{i^*}}$  can be increased as long as  $-\bar{a}_{i^*,j} w_{B_{i^*}} \leq \bar{w}_j$  is satisfied for all  $j \in N$ . If  $x_{B_{i^*}} < 0$  and  $\bar{a}_{i^*,j} \geq 0$  for all  $j \in N$ , then dual problem is unbounded and the primal problem is infeasible. The potentially binding constraints are those with  $\bar{a}_{i^*,j} < 0$ . Therefore, there is a corresponding dual ratio test which is

$$q := \operatorname{argmin} \{-(\bar{w}_j / \bar{a}_{i^*,j}) \mid \bar{a}_{i^*,j} < 0, j \in N\} \quad (6.17)$$

and the dual step length is

$$\alpha_D := (\bar{w}_q / \bar{a}_{i^*,q}). \quad (6.18)$$

If the nonbasic dual variable  $w_{B_{i^*}}$  is increased from 0 to  $-\alpha_D$  how do the basic dual variables change? This is given in (6.15) and (6.16). Denote the current dual solution by

$$\bar{u} = A_B^{-\top} c_B - A_B^{-\top} \bar{w}_B, \quad \bar{w}_N = c_N - A_N^T A_B^{-\top} c_B + A_N^T A_B^{-\top} \bar{w}_B, \quad \bar{w}_B = 0.$$

Then if  $\bar{w}_{B_{i^*}}$  is increased from 0 to  $-\alpha_D$  the new dual solution is

$$\hat{u} = \bar{u} + \alpha_D A_B^{-\top} e^{i^*} \quad (6.19)$$

$$\hat{w}_N = \bar{w}_N - \alpha_D A_N^T A_B^{-\top} e^{i^*} \quad (6.20)$$

$$\hat{w}_{B_{i^*}} = -\alpha_D. \quad (6.21)$$

Now refer back to the description of Algorithm 5.9, the revised primal simplex algorithm, in Chapter 5. The dual adjustments are identical. The only difference between primal simplex and dual simplex is that in primal simplex we maintain primal feasibility and work towards dual feasibility. The primal ratio test is used to guarantee that the next primal iterate is feasible. In dual simplex, we have a dual feasible solution and work to primal feasibility. The dual ratio test guarantees that the next dual iterate is dual feasible.

### Algorithm 6.9 (Dual Simplex)

**Step 1: (Initialization)** Initialize with a basis matrix  $A_B$  such that  $\bar{w}_N \geq 0$ . Initialize the primal and dual solution

$$\bar{x}_B \leftarrow A_B^{-1}b, \quad \bar{z}_0 \leftarrow c_B^\top A_B^{-1}b, \quad \bar{u}^\top \leftarrow c_B^\top A_B^{-1}, \quad \bar{w}_N^\top \leftarrow c_N^\top - \bar{u}^\top A_N.$$

**Step 2: (Pivot Row Selection)** If  $\bar{x}_B \geq 0$ , stop, the current solution is optimal; else, select  $i^* \in \{1, \dots, m\}$  such that  $\bar{x}_{B,i^*} < 0$ .

**Step 3: (Row Update)** Find row  $i^*$  of  $A_B^{-1}$  through the BTRAN

$$(z^{i^*})^\top \leftarrow (e^{i^*})^\top A_B^{-1}$$

and compute  $(z^{i^*})^\top A_N$  which is row  $i^*$  of  $A_B^{-1}A_N$ .

**Step 4: (Dual Minimum Ratio Test and Pivot Column Selection)** If  $\bar{a}_{i^*,j} \geq 0$  for all  $j \in N$  stop, the linear program is infeasible. Otherwise perform the dual simplex minimum ratio test

$$q \leftarrow \operatorname{argmin} \{-(\bar{w}_j / \bar{a}_{i^*,j}) \mid \bar{a}_{i^*,j} < 0, j \in N\}.$$

Update the primal and dual step lengths.

$$\alpha_P \leftarrow \bar{x}_{B,i^*} / \bar{a}_{i^*,q}, \quad \alpha_D \leftarrow \bar{w}_q / \bar{a}_{i^*,q}$$

**Step 5: (Update the Pivot Column)** Calculate the updated column  $\bar{a}^q$  through the FTRAN

$$\bar{a}^q \leftarrow A_B^{-1}a^q.$$

**Step 6: (Update the Primal and Dual Solution, Basis and Basis Inverse)**

$$\bar{x}_{B,i} \leftarrow \bar{x}_{B,i} - \alpha_P \bar{a}_{iq}, \quad i = 1, \dots, m, \quad \bar{x}_q \leftarrow \alpha_P, \quad \bar{z}_0 \leftarrow \bar{z}_0 + \alpha_P \bar{w}_q$$

$$\bar{u}^\top \leftarrow \bar{u}^\top + \alpha_D(z^{i^*})^\top, \quad \bar{w}_N^\top \leftarrow \bar{w}_N^\top - \alpha_D(z^{i^*})^\top A_N, \quad \bar{w}_{B_{i^*}} \leftarrow -\alpha_D$$

*Update the index of basic variables.*

$$B \leftarrow (B \setminus \{B_{i^*}\}) \cup \{q\}, \quad N \leftarrow (N \setminus \{q\}) \cup \{B_{i^*}\}$$

*and update the basis inverse*

$$A_B^{-1} \leftarrow \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*q}} \right) A_B^{-1}.$$

*Go to Step 2.*

At this point the reader may wish to refer back to Algorithm 5.9, the revised simplex algorithm, and compare it with dual simplex. For both algorithms, the primal and dual updates are identical. So are the primal and dual step sizes and the basis inverse. The difference in the two methods is the ratio test, and the order in which the FTRAN and BTRAN operations are made. There are a number of cases where dual simplex may be preferred over primal simplex.

1. A constraint is added. Later in Chapter 9 variables are forced to take on integer values by adding constraints to a linear program which has already been optimized. Adding a constraint to an optimal tableau is illustrated next in Example 6.10. Adding a constraint to an optimal tableau may destroy primal feasibility, but the tableau remains dual optimal making dual simplex a logical choice to use for reoptimizing.
2. A change is made to the right hand side vector. In the Section 6.2 ranges were calculated for each right hand side so that any change in the right hand side within the allowable increase or decrease maintains primal feasibility and hence primal optimality. In many cases it is important to know the optimal solution for changes outside the allowable increase and decrease. Obviously one could resolve the linear program for the new right hand side. However, to start all over from scratch is not necessary. The current tableau with a dual feasible solution can be used as a starting point for the dual simplex algorithm.
3. Dual simplex might be faster than primal simplex. We defer this point until later.

**Example 6.10** Refer back to Example 5.2 in Chapter 5. The optimal tableau is given below.

$$\begin{array}{rccccccccc} z_0 & & -2x_5 & -x_6 & = & -22 \\ x_3 & & +2x_5 & -x_6 & = & 9 \\ x_2 & & +2x_5 & -x_6 & = & 10 \\ x_1 & & +2x_5 & -2x_6 & = & 4 \\ x_4 & & +4x_5 & -3x_6 & = & 12 \end{array}$$

Add the constraint  $-3x_1 + 4x_2 + x_7 = 2$  to the problem and assume that the original right hand side vector is changed from  $b^T = [1, 2, 8, 6]$  to  $b^T = [25, 2, 8, 6]$ . Then

$$A_B^{-1}b = \begin{bmatrix} -1 & 0 & 2 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 2 & -2 \\ 0 & -1 & 4 & -3 \end{bmatrix} \begin{bmatrix} 25 \\ 2 \\ 8 \\ 6 \end{bmatrix} = \begin{bmatrix} -15 \\ 10 \\ 4 \\ 12 \end{bmatrix}$$

and the modified tableau associated with the old optimal basis is

$$\begin{array}{rccccccccc} z_0 & & -2x_5 & -x_6 & & = & -22 \\ x_3 & & +2x_5 & -x_6 & & = & -15 \\ x_2 & & +2x_5 & -x_6 & & = & 10 \\ x_1 & & +2x_5 & -2x_6 & & = & 4 \\ x_4 & & +4x_5 & -3x_6 & & = & 12 \\ +4x_2 & -3x_1 & & & & +x_7 & = & 2 \end{array}$$

Assume  $x_7$  is basic and pivot on  $x_1$  and  $x_2$  so the tableau contains an identity matrix.

$$\begin{array}{rccccccccc} z_0 & & -2x_5 & -x_6 & & = & -22 \\ x_3 & & +2x_5 & -x_6 & & = & -15 \\ x_2 & & +2x_5 & -x_6 & & = & 10 \\ x_1 & & +2x_5 & -2x_6 & & = & 4 \\ x_4 & & +4x_5 & -3x_6 & & = & 12 \\ x_7 & -2x_5 & -2x_6 & & & = & -26 \end{array}$$

This tableau is dual feasible and satisfies complementary slackness. However, primal feasibility is violated because variable  $x_7 = -26$  and  $x_3 = -15$ . Since there is a feasible dual solution apply dual simplex.

**Step 2: (Pivot Row Selection)** Variable  $x_7$  violates primal feasibility by the largest amount so select it as the pivot row.

**Step 3: (Row Update)** Since we are working with the tableau we have the updated row which is the last row of the tableau.

**Step 4: (Dual Minimum Ratio Test and Pivot Column Selection)**

$$\min\{-\bar{w}_5/\bar{a}_{55}, -\bar{w}_6/\bar{a}_{56}\} = \{-(2/-2), -(1/-2)\} = 1/2$$

The minimum ratio is for variable  $x_6$ , the dual step length is  $-1/2$  and the primal step length is  $-26/-2$ . Variable  $x_6$  pivots into the basis and variable  $x_7$  out of the basis.

**Step 5: (Update the Pivot Column)** Since we are working with the tableau we have the updated column for variable  $x_6$ .

**Step 6: (Update the Primal and Dual Solutions, Basis and Basis Inverse)** Bring  $x_6$  into the basis and update the tableau.

$$\begin{array}{rccccc}
 z_0 & -\frac{1}{2}x_7 & -x_5 & & = & -9 \\
 x_3 & -\frac{1}{2}x_7 & +3x_5 & & = & -2 \\
 x_2 & -\frac{1}{2}x_7 & +3x_5 & & = & 23 \\
 x_1 & -x_7 & +4x_5 & & = & 30 \\
 x_4 & -\frac{3}{2}x_7 & +7x_5 & & = & 51 \\
 & -\frac{1}{2}x_7 & +x_5 & +x_6 & = & 13
 \end{array}$$

Now variable  $x_3$  has a negative value so the row in which  $x_3$  is basic is the pivot row and the example continues as in the previous pivot.

It is also possible that it is preferable to solve a linear program “from scratch” with dual simplex rather than primal simplex. In the previous chapter, we pointed out that primal degeneracy is a problem for primal simplex. If a problem is highly primal degenerate, but not dual degenerate then dual simplex might perform much better than primal simplex. Also, consider the linear knapsack problem

$$\begin{aligned}
 \min \quad & \sum_{j=1}^n c_j x_j \\
 \text{subject to} \quad & \sum_{j=1}^n a_j x_j - x_{n+1} = b \\
 & x_j \geq 0, \quad j = 1, \dots, n+1.
 \end{aligned}$$

Assume all  $c_j, a_j \geq 0$ . Start with the primal infeasible solution  $x_{n+1} = -b$ ,  $x_j = 0$ . We leave it as an exercise to prove that dual simplex algorithm finds the optimal solution in one pivot. It simply finds the primal variable with the best bang for buck ratio and pivots it in. Bixby [55] reports computational

results comparing primal simplex versus dual simplex. On a test set of eight large problems, the dual simplex algorithm was faster on three of the problems. Later in Subsection 6.6.4 we show that the dual simplex algorithm requires less work than the primal simplex algorithm when using steepest edge pricing.

The dual simplex algorithm begins with a basic feasible dual solution and complementary primal solution which is not necessarily feasible. The *primal-dual algorithm* is similar to dual simplex in that it also assumes a feasible (but not necessarily basic) solution to the dual problem and works to find a complementary feasible primal solution. This algorithm has been particularly effective in solving network flow problems discussed later in Chapter 14. See Bazaraa, Jarvis, and Sherali [43] for a description of the primal-dual algorithm.

## 6.4 SIMPLE UPPER BOUNDS AND SPECIAL STRUCTURE

The linear program in standard form is  $\min\{c^T x \mid Ax = b, x \geq 0\}$ . In many applications there are *simple upper bounds* on the variables. That is,  $x_j \leq h_j$  for  $j = 1, \dots, n$ . For example,  $x_j$  might be the flow on an arc in a network and there is a restriction on the arc flow. See Chapter 14. By adding a slack variable  $v_j$ , each simple upper bound constraint is converted into the equality constraint  $x_j + v_j = h_j$ . There is a problem with treating the simple upper bounds as regular constraints. In many cases the number of variables is large relative to the number of constraints and adding the constraints  $x_j + v_j = h_j$  results in a huge increase in the size of the basis matrix. Since this basis matrix is inverted at each pivot, this greatly increases the computational effort. In this section we show how to treat simple upper bounds on the variables without increasing the size of the basis matrix. We also allow for arbitrary *simple lower bounds*  $l_j$  on the variables which differ from zero. Assume that the program is

$$\begin{aligned} & \min c^T x \\ \text{s.t. } & Ax = b \\ & x \leq h \\ & x \geq l \end{aligned}$$

and that  $A$  has rank  $m$ . Again, define the columns of  $A$  indexed by the ordered set  $B$  of cardinality,  $m$ , as a basis if the columns are linearly independent. The solution  $\bar{x}_B = A_B^{-1}b$ ,  $\bar{x}_j = l_j$  or  $\bar{x}_j = h_j$  for all  $j \in N$  is a basic feasible solution if  $l_{B_i} \leq x_{B_i} \leq h_{B_i}$  for  $i = 1, \dots, m$ . Thus, a basic feasible solution is one in

which all of the nonbasic variables are at their upper or lower bounds, and all the basic variables satisfy the lower and upper bound constraints. With this extension basic feasible solutions still correspond to extreme points.

**Proposition 6.11 (Basic Feasible Solution)** *The vector  $\bar{x} \in \mathbb{R}^n$  is a basic feasible solution of the system  $Ax = b$ ,  $x \leq h$ ,  $x \geq l$  if and only if it is an extreme point of  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \leq h, x \geq l\}$ .*

Given a basis  $B$ , the projection of the linear program onto the space of nonbasic variables is

$$\begin{aligned} z_0 - \bar{w}_N^\top x_N &= c_B^\top A_B^{-1} b \\ A_B^{-1} A_N x_N &\leq l_B + A_B^{-1} b \\ A_B^{-1} A_N x_N &\geq A_B^{-1} b - h_B \\ x_N &\leq h_N \\ x_N &\geq l_N \end{aligned}$$

where again  $\bar{w}_N^\top = c_N^\top - c_B^\top A_B^{-1} A_N$ .

Assume the basic solution  $(\bar{x}_B, \bar{x}_N)$  corresponding to  $B$  is feasible. How can this solution be improved? If  $\bar{x}_j = l_j$  and  $\bar{w}_j < 0$  for  $j \in N$ , then the basic feasible solution is improved by increasing variable  $x_j$  from its lower bound. If  $\bar{x}_j = h_j$  and  $\bar{w}_j > 0$  for  $j \in N$ , then the basic feasible solution is improved by decreasing variable  $x_j$  from its upper bound. If, for all  $j \in N$ ,  $\bar{x}_j = l_j$  implies  $\bar{w}_j \geq 0$  and  $\bar{x}_j = h_j$  implies  $\bar{w}_j \leq 0$ , then the basic feasible solution  $(\bar{x}_B, \bar{x}_N)$  is also optimal. This result is analogous to Proposition 5.1. Given this observation, only two simple modifications to the revised simplex algorithm are necessary. The first modification is in Step 2, selecting the entering variable, and the second modification is in Step 4, the minimum ratio test. When selecting an entering variable, select a nonbasic variable  $x_q$  at its lower bound with negative reduced cost, or a nonbasic variable  $x_q$  at its upper bound with positive reduced cost. If none exist, terminate with the optimal solution.

The minimum ratio test is altered as follows. First, consider the case where the entering variable  $x_q$  is at its lower bound  $l_q$ . This is similar to what we have done so far with a lower bound of zero. However, when there are upper and lower bounds on the basic variables, increasing the nonbasic variable  $x_q$  from its lower bound of  $l_q$  may force a basic variable to its upper or lower bound. Recall

$$\begin{bmatrix} \hat{x}_B \\ \hat{x}_N \end{bmatrix} = \begin{bmatrix} \bar{x}_B \\ \bar{x}_N \end{bmatrix} + \alpha_P \begin{bmatrix} \Delta x_B \\ \Delta x_N \end{bmatrix}$$

where

$$\Delta x_B = \begin{bmatrix} -\bar{a}_{1q} \\ -\bar{a}_{2q} \\ \vdots \\ -\bar{a}_{mq} \end{bmatrix}$$

and  $\Delta x_N$  is an  $(n-m)$  component unit vector with the 1.0 in the component corresponding to variable  $x_q$ . If  $\bar{a}_{iq} > 0$ , increasing  $x_q$  from its lower bound may result in  $x_{B_i}$  hitting its lower bound. Then  $\bar{x}_{B_i} - \alpha_P \bar{a}_{iq} \geq l_{B_i}$ , or

$$\alpha_P \leq \theta_1 := \min \left\{ \frac{\bar{x}_{B_i} - l_{B_i}}{\bar{a}_{iq}} \mid \bar{a}_{iq} > 0, i = 1, \dots, m \right\}.$$

If  $\bar{a}_{iq} < 0$ , increasing  $x_q$  from its lower bound may result in  $x_{B_i}$  hitting its upper bound. Then  $\bar{x}_{B_i} - \alpha_P \bar{a}_{iq} \leq h_{B_i}$ , or

$$\alpha_P \leq \theta_2 := \min \left\{ \frac{\bar{x}_{B_i} - h_{B_i}}{\bar{a}_{iq}} \mid \bar{a}_{iq} < 0, i = 1, \dots, m \right\}.$$

Also,  $x_q$  cannot exceed its upper bound so  $\alpha_P \leq h_q - l_q$ . Therefore,

$$\alpha_P = \min \{(h_q - l_q), \theta_1, \theta_2\}. \quad (6.22)$$

Compare the new value  $\alpha_P$  given in (6.22) with the value for  $\alpha_P$  given in the original ratio test (5.12). If all the upper bounds are infinity and the lower bounds are zero, then  $\theta_2 = \infty$  and  $\theta_1$  reduces to (5.12). When  $\alpha_P = (h_q - l_q)$  the nonbasic variable  $x_q$  moves from its lower bound to its upper bound and there is no change of basis. The objective function value does change from  $\bar{z}_0$  to  $\bar{z}_0 + (h_q - l_q)\bar{w}_q$ .

Now consider the case when the nonbasic variable  $x_q$  is decreased from its upper bound. In this case

$$\Delta x_B = \begin{bmatrix} \bar{a}_{1q} \\ \bar{a}_{2q} \\ \vdots \\ \bar{a}_{mq} \end{bmatrix}$$

and  $\Delta x_N$  is the negative of the  $(n-m)$  component unit vector with the 1.0 in the component corresponding to variable  $x_q$ . If  $\bar{a}_{iq} > 0$ , decreasing  $x_q$  from its upper bound may result in  $x_{B_i}$  hitting its upper bound. Then  $\bar{x}_{B_i} + \alpha_P \bar{a}_{iq} \leq h_{B_i}$ , or

$$\alpha_P \leq \theta_3 := \min \left\{ \frac{h_{B_i} - \bar{x}_{B_i}}{\bar{a}_{iq}} \mid \bar{a}_{iq} > 0, i = 1, \dots, m \right\}.$$

If  $\bar{a}_{iq} < 0$ , decreasing  $x_q$  from its upper bound may result in  $x_{B_i}$  hitting its lower bound. Then  $\bar{x}_{B_i} + \alpha_P \bar{a}_{iq} \geq l_{B_i}$ , or

$$\alpha_P \leq \theta_4 := \min \left\{ \frac{l_{B_i} - \bar{x}_{B_i}}{\bar{a}_{iq}} \mid \bar{a}_{iq} < 0, i = 1, \dots, m \right\}.$$

Also,  $x_q$  cannot go below its lower bound so  $\alpha_P \leq h_q - l_q$ . Therefore,

$$\alpha_P = \min \{(h_q - l_q), \theta_3, \theta_4\} \quad (6.23)$$

If all the upper bounds are infinity this case is vacuous. If  $\alpha_P = (h_q - l_q)$  the nonbasic variable  $x_q$  moves from its upper bound to its lower bound and there is no change of basis. The objective function value again changes from  $\bar{z}_0$  to  $\bar{z}_0 + (h_q - l_q)\bar{w}_q$ .

We have shown how to handle simple and upper lower bound constraints without increasing the size of the working basis. There are other important structures for which this is also possible. Dantzig and Van Slyke [116] show how to handle *generalized upper bound* (GUB) constraints without increasing the size of the basis. These are constraints with the structure

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

Schrage [400, 401] has extended this to *variable upper bound* constraints with the structure

$$x_{ij} \leq y_i, \quad j = 1, \dots, m_i, \quad i = 1, \dots, n$$

and *generalized variable upper bound* constraints with the structure

$$\sum_{j=1}^{m_i} x_{ij} \leq y_i, \quad i = 1, \dots, n.$$

See also Todd [424]. The variable upper bound constraints are often present in location models. See, for example, the simple plant location model introduced in Subsection 1.3.5 in Chapter 1. For work on trying to find these special structure constraints in arbitrary constraint matrices see Gunawardane, Hoff, and [219].

## 6.5 FINDING A STARTING BASIS

The primal simplex algorithm requires a starting basic feasible solution. If a known feasible, but not necessarily basic, solution  $\bar{x}$  exists for the problem then it can be *crashed* to find a basic feasible solution  $\hat{x}$  such that  $c^T \hat{x} \leq c^T \bar{x}$  (minimization assumed). See the exercises. If a feasible primal solution is not known, there are two basic methods for finding a feasible solution. Both methods begin with a *Phase I* problem which is a modification of the original problem. The solution of the Phase I problem produces a primal basic feasible solution and *Phase II* is begun. The Phase II problem is the original linear program.

The first strategy is to introduce *artificial variables* into a Phase I problem and begin with a basic feasible solution consisting of the artificial variables. Create the Phase I linear program by replacing the original linear program  $\min\{c^T x \mid Ax = b, x \geq 0\}$  with the linear program  $\min\{e^T x^a \mid Ax + Ix^a = b, x \geq 0\}$  where  $x^a$  is an  $m$  vector of *artificial variables* and  $e$  is an  $m$  vector with each component 1.0. Without loss, assume  $b_i \geq 0$  for  $i = 1, \dots, m$ . Why is nonnegativity of the right hand side justified? A basic feasible solution to the new system is  $x_i^a = b_i$  for  $i = 1, \dots, m$ . If the system  $Ax = b, x \geq 0$  has a feasible solution then simplex applied to the Phase I problem will terminate in a finite number of pivots (assuming appropriate pivoting rules are used to prevent cycling) with an optimal objective function value of zero and a feasible solution to the original problem.

However, there is still a potential problem even if the optimal solution value to the Phase I problem is zero. The optimal solution found by simplex could be degenerate. That is, an artificial variable  $x_k^a$  is equal to zero, but is part of the optimal basis. Assume without loss, that artificial variable  $x_k^a$  is the  $k$ th basic variable. There are two cases to consider.

1. There is a nonbasic variable  $x_l$  and the updated tableau coefficient  $\bar{a}_{kl}$  of  $x_l$  is nonzero. In this case variable  $x_l$  is pivoted into the basis and variable  $x_k^a$  is pivoted out of the basis.
2. In the updated tableau,  $\bar{a}_{kl} = 0$  for every nonbasic variable  $x_l$ . In this case, row  $k$  is redundant and can be deleted.

By repeating the logic used in these two cases it is possible to find a basic feasible solution which does not contain any artificial variables. After solving the Phase I problem with artificial variables, simplex enters Phase II which is the original linear program with the original objective function.

Another approach described in Greenberg [210] (who credits Dennis Rarick) is to start with a solution which is basic, but infeasible and then minimize the

*sum of infeasibilities.* For simplicity, assume simple lower bounds of zero and simple upper bounds of  $\infty$  on all variables. Define

$$\theta_j(x) := \begin{cases} 0, & x_j \geq 0 \\ -x_j, & x_j < 0 \end{cases}$$

and solve the Phase I problem

$$\begin{aligned} \min \quad & \sum_{j=1}^n \theta_j(x) \\ \text{s.t.} \quad & Ax = b. \end{aligned}$$

This is not a linear program since the objective function, the sum of infeasibilities, is a piecewise linear function of  $x$ . Nevertheless, the concept of pivoting variables in and out of the basis still applies. Consider the following example.

### Example 6.12

$$\begin{array}{rclcl} x_1 & +2x_5 & = & -5 \\ x_2 & +4x_5 & = & 8 \\ x_3 & -5x_5 & = & -12 \\ x_4 & -3x_5 & = & -3 \end{array}$$

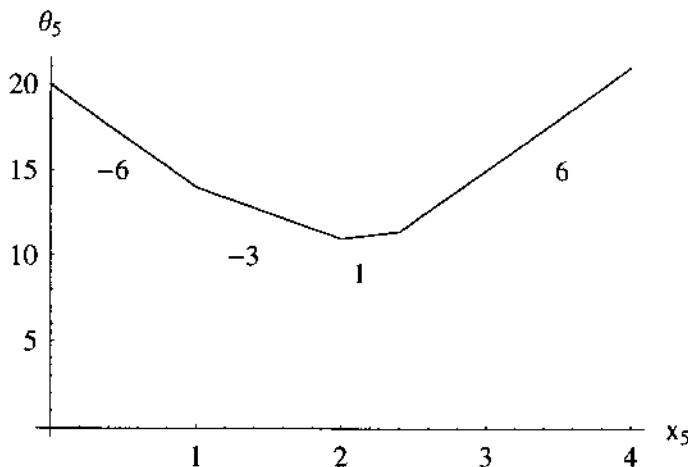
The basis is  $B = \{1, 2, 3, 4\}$  and

$$\bar{x} = (-5, 8, -12, -3), \theta_1(\bar{x}) = 5, \theta_2(\bar{x}) = 0, \theta_3(\bar{x}) = 12, \theta_4(\bar{x}) = 3, \theta_5(\bar{x}) = 0.$$

The sum of infeasibilities is 20. Consider the effect of pivoting  $x_5$  into the basis. If  $x_5 = \epsilon \leq 1$  then basic variable  $x_1$  becomes more infeasible by  $2\epsilon$ , basic variable  $x_2$  is feasible and remains feasible, basic variable  $x_3$  moves towards its lower bound by  $5\epsilon$  units and basic variable  $x_4$  moves towards its lower bound by  $3\epsilon$  units. Thus, the sum of infeasibilities function

$$\sum_{j=1}^5 \theta_j(\bar{x} + \epsilon e^5)$$

has a slope of  $-6$  and  $\epsilon$  should be increased as long as the slope remains negative. If  $\epsilon$  is increased to 1.0, then basic variable  $x_4$  becomes feasible and slope changes to  $-3$ . If  $\epsilon$  is increased to 2.0 the slope now becomes  $+1$ . Why? A plot of the sum of infeasibilities is given below in Figure 6.1. The slope of each part of the piecewise linear curve is displayed in the figure.

**Figure 6.1** Sum of Infeasibilities for Example 6.12

Given a large problem, how does one select the entering nonbasic variable in an effort to reduce the sum of infeasibilities? Just as with the artificial variable technique, a Phase I objective function is used. Assuming the nonbasic variables are at their upper or lower bounds, a basic variable is assigned a cost of  $-1$  if it is below its lower bound, a cost of  $0$  if it is feasible and a cost of  $+1$  if it is above its upper bound. Minimizing the sum of infeasibilities is used in a Phase I procedure in LINDO. A good formal description of the method is in Greenberg [210].

Regardless of which Phase I method is used, it is desirable to pick an initial basis that requires as few artificial variables as possible, or makes the sum of infeasibilities as small as possible. For example, if a constraint is in  $\leq$  form with a nonnegative right hand side, then it is possible to use the slack variable for that constraint rather than use an artificial variable. In fact, if the linear program is originally of the form  $Ax \leq b$  and  $b \geq 0$ , then a starting basic solution is given by adding slack variables to the matrix and letting slack variable  $i$  equal  $b_i$ . When minimizing the sum of infeasibilities it is desirable to select basic variables where there is a very large difference between the upper and lower bounds in order to allow nonbasic variables to pivot in as much as possible before being blocked. See Bixby [54] for a description of how an initial basis is found for CPLEX.

Finally, it is common to use a *composite* objective function which has both original variables possibly at a negative value and artificial variables at a positive value. In this case, if  $x_k^a$  is an artificial variable, it is never allowed to go negative and its contribution to the sum of infeasibilities is  $\theta_k(x_k^a) = x_k^a$ . See Wolfe [458].

## 6.6 PIVOT COLUMN SELECTION

In Step 2 of the revised simplex algorithm it is necessary to select a new column to enter the basis. The rule used thus far is the Dantzig rule of selecting the variable with the most negative reduced cost. This is not necessarily the most effective strategy.

### 6.6.1 Partial Pricing

In practice, a linear program often contains far more columns than rows. Rather than calculate the reduced cost of every variable at every pivot, with partial pricing a subset of the columns are selected. The linear program is then optimized over this subset of columns. When all the columns in this subset price out nonnegative another subset of columns is selected. This strategy was motivated by computing technology. When simplex codes were first developed it was impossible to store the entire constraint matrix in core. Because reading data from secondary storage was costly, partial pricing provided a substantial savings in secondary storage to central memory transfer time.

### 6.6.2 Multiple Pricing

Multiple pricing is like partial pricing in that only a subset of the columns are used at each iteration. The reduced cost of variable  $x_q$  gives the per unit decrease in the objective function value as  $x_q$  enters the basis. However, the reduced cost provides no information on how much  $x_q$  can be increased. This information is provided in the minimum ratio test and requires  $\bar{a}^q$ . Ideally, one would like to find the nonbasic variable which maximizes  $-\alpha_P \bar{w}_q$ . Unfortunately, this requires calculating  $A_B^{-1} A_N$  which is very cumbersome. With multiple pricing,  $\bar{a}^q$  is kept in main memory for a small subset of columns and the maximum value of  $\alpha_P \bar{w}_q$  is found for this subset of columns. At each iteration, as the basis changes from  $B$  to  $\tilde{B}$ , the new updated columns are calculated

from the ones at the previous iteration by multiplying through by the new eta matrix, i.e.  $\hat{a}^q = E\bar{a}^q$ .

### 6.6.3 Primal Steepest Edge

Selecting the pivot column based on the most negative reduced cost has an interesting interpretation in terms of the linear program in the projected space of the nonbasic variables. Repeating (5.7) - (5.9) from the previous chapter, the projected linear program in the space of nonbasic variables is

$$\begin{aligned} z_0 - \bar{w}_N^\top x_N &= c_B^\top A_B^{-1} b \\ A_B^{-1} A_N x_N &\leq A_B^{-1} b \\ x_N &\geq 0. \end{aligned}$$

Given a differentiable function  $f(x)$ , from  $\mathbb{R}^n \rightarrow \mathbb{R}^1$ , the directional derivative of  $f(x)$  in direction  $d \in \mathbb{R}^n$  is the inner product  $\nabla f(x)(d/\|d\|)$ . Thus, if  $f(x_N) = \bar{w}_N^\top x_N$ , selecting

$$q = \operatorname{argmin} \{\bar{w}_j \mid \bar{w}_j < 0, j \in N\}$$

and increasing  $x_q$ , results in a move in the direction with the minimum directional derivative. But this is in the space of nonbasic variables only. What about the space of both nonbasic and basic variables? In this case,

$$f(x) = c^\top x = c_B^\top x_B + c_N^\top x_N.$$

When the nonbasic variable  $\bar{x}_q$  is increased from 0, the new solution  $\hat{x}$  defined from the old solution  $\bar{x}$  is

$$\begin{bmatrix} \hat{x}_B \\ \hat{x}_N \end{bmatrix} = \begin{bmatrix} \bar{x}_B \\ \bar{x}_N \end{bmatrix} + \alpha_P \begin{bmatrix} \Delta x_B \\ \Delta x_N \end{bmatrix}$$

so the direction of change is

$$d = \begin{bmatrix} \Delta x_B \\ \Delta x_N \end{bmatrix} \quad (6.24)$$

where  $\Delta x_B = -\bar{a}^q$  and  $\Delta x_N$  is an  $(n-m)$  component vector with the 1.0 in the component corresponding to  $x_q$ . Then the directional derivative of  $f(x)$  in the direction  $d$  is

$$\begin{aligned} \nabla f(x)(d/\|d\|) &= (c_B^\top \Delta x_B + c_N^\top \Delta x_N)/\|d\| \\ &= (c_q - c_B^\top \bar{a}^q)/\|d\| \\ &= (c_q - c_B^\top A_B^{-1} a^q)/\|d\| \\ &= \bar{w}_j/\|d\|. \end{aligned}$$

Now what is  $\|d\|$ ? From (6.24),

$$\begin{aligned}\|d\|^2 &= \|\Delta x_B\|^2 + 1 \\ &= \|\bar{a}^q\|^2 + 1 \\ &= (a^q)^\top A_B^{-\top} A_B^{-1} a^q + 1.\end{aligned}$$

Therefore, in order to find which nonbasic variable increase results in the largest directional derivative, i.e. the *steepest descent*, in the space of both nonbasic variables and basic variables it is necessary to calculate

$$\nu_j := (a^j)^\top A_B^{-\top} A_B^{-1} a^j \quad (6.25)$$

for each  $j \in N$ . Calculating  $\nu_j$  using (6.25) requires an FTRAN and an inner product calculation for every nonbasic variable, which is prohibitive. Fortunately, Goldfarb and Reid [189] give a much more efficient way to find the  $\nu_j$ . Increasing a nonbasic variable from zero until a basic variable pivots out of the basis corresponds to moving from one extreme point of a polyhedron to another along an edge of the polyhedron. Hence, this approach is called *steepest edge pricing*.

Recall, that in the row update step (Step 5) of revised simplex we calculate

$$z^{i^*} = (e^{i^*})^\top A_B^{-1}$$

with a BTRAN operation. In addition define,

$$y^q := A_B^{-\top} \bar{a}^q. \quad (6.26)$$

This also requires a BTRAN. Given  $\nu_j = (a^j)^\top A_B^{-\top} A_B^{-1} a^j$  from the previous simplex iteration, we want to calculate  $\hat{\nu}_j = (a^j)^\top A_{\hat{B}}^{-\top} A_{\hat{B}}^{-1} a^j$  for the new basis  $\hat{B}$ . This is done most efficiently using the rank one update from (5.23) in Chapter 5. The rank one update is

$$A_{\hat{B}}^{-1} = EA_B^{-1} = \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*q}} \right) A_B^{-1}.$$

Then the update of  $\hat{\nu}_j$  is

$$\begin{aligned}\hat{\nu}_j &= (a^j)^\top A_{\hat{B}}^{-\top} A_{\hat{B}}^{-1} a^j = (a^j)^\top A_B^{-\top} E^\top EA_B^{-1} a^j \\ &= (a^j)^\top A_B^{-\top} \left( I - \frac{e^{i^*}(\bar{a}^q - e^{i^*})^\top}{\bar{a}_{i^*q}} \right) \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^*q}} \right) A_B^{-1} a^j \\ &= \left( (a^j)^\top A_B^{-\top} - \frac{(a^j)^\top (z^{i^*}) (\bar{a}^q - e^{i^*})^\top}{\bar{a}_{i^*q}} \right) \left( A_B^{-1} a^j - \frac{(\bar{a}^q - e^{i^*})(z^{i^*})^\top a^j}{\bar{a}_{i^*q}} \right) \\ &= \nu_j - 2 \frac{(a^j)^\top (y^q - z^{i^*}) (z^{i^*})^\top a^j}{\bar{a}_{i^*q}} + \frac{((a^j)^\top z^{i^*})^2 (\bar{a}^q - e^{i^*})^\top (\bar{a}^q - e^{i^*})}{(\bar{a}_{i^*q})^2}\end{aligned}$$

How much more work is involved in calculating the steepest ascent in the variable space  $(x_B, x_N)$  than in the variable space  $x_N$ ? Both  $z^{i^*}$  and  $y^q$  must be calculated. These calculations require a BTRAN. However,  $z^{i^*}$  is already calculated in Step 5 of the revised simplex, so only one extra BTRAN is required. In addition, the inner products  $(a^j)^\top y^k$  and  $(a^j)^\top z^{i^*}$  are required for each  $j \in \hat{N}$ .

Goldfarb and Reid [189] demonstrate the computational viability of this method. There are other approaches such as the Devex method of Harris [223], which are similar in spirit, in that they amount to approximations of steepest edge pricing. See also Crowder and Hattingh [102].

### 6.6.4 Dual Steepest Edge

A similar analysis is appropriate for the dual simplex algorithm. In Step 2 of the dual simplex algorithm, one way to select the pivot row is to use the dual of the Dantzig rule and select the most negative  $\bar{x}_{B_i}$ . However, the steepest edge philosophy can also be applied. If the primal linear program is  $\min \{c^\top x \mid Ax = b, x \geq 0\}$  there are several dual problems. We consider the dual problem without slack variables and work with  $\max \{b^\top u \mid A^\top u \leq c\}$ . See Forrest and Goldfarb [153]. In the dual simplex algorithm, if row  $k$  is the pivot row, the dual variable update is  $\hat{u} = \bar{u} + \alpha_D z^k$  where  $z^k = A_B^{-\top} e^k$ . Then the directional derivative in direction  $z^k$  is

$$b^\top z^k / \|z^k\| = b^\top A_B^{-\top} e^k / \|z^k\| = \bar{x}_{B_k} / \|z^k\|.$$

If the new basis  $\hat{B}$  is  $B \setminus B_{i^*} \cup \{q\}$  and  $\hat{\eta}_k = \|z^k\|^2$ , then

$$\begin{aligned} \hat{\eta}_k &= \|A_{\hat{B}}^{-\top} e^k\|^2 = (e^k)^\top A_{\hat{B}}^{-1} A_{\hat{B}}^{-\top} e^k \\ &= (e^k)^\top \left( I - \frac{(\bar{a}^q - e^{i^*})(e^{i^*})^\top}{\bar{a}_{i^* q}} \right) A_B^{-1} A_B^{-\top} \left( I - \frac{(e^{i^*})(\bar{a}^q - e^{i^*})^\top}{\bar{a}_{i^* q}} \right) e^k \\ &= (e^k)^\top \left( A_B^{-1} - \frac{(\bar{a}^q - e^{i^*})(z^{i^*})^\top}{\bar{a}_{i^* q}} \right) \left( A_B^{-\top} - \frac{z^{i^*} (\bar{a}^q - e^{i^*})^\top}{\bar{a}_{i^* q}} \right) e^k \\ &= (e^k)^\top A_B^{-1} A_B^{-\top} e^k - \frac{2(e^k)^\top (\bar{a}^q - e^{i^*})(z^{i^*})^\top A_B^{-\top} e^k}{\bar{a}_{i^* q}} \\ &\quad + \frac{(e^k)^\top (\bar{a}^q - e^{i^*})(z^{i^*})^\top z^{i^*} (\bar{a}^q - e^{i^*})^\top e^k}{\bar{a}_{i^* q}^2} \\ &= \eta_k - 2 \left( \frac{\bar{a}_{kq}}{\bar{a}_{i^* q}} \right) (z^{i^*})^\top A_B^{-\top} e^k + \left( \frac{\bar{a}_{kq}^2}{\bar{a}_{i^* q}^2} \right) (z^{i^*})^\top z^{i^*} \end{aligned}$$

The majority of the work in calculating  $\eta_k$  is the FTRAN operation  $A_B^{-1}z^{i^*} = (z^{i^*})^\top A_B^{-\top}$ . The only arithmetic operations which must be done for every row  $k$  are calculating one ratio ( $\bar{a}_{kq}/\bar{a}_{i^*q}$ ) and then squaring the result. The inner product  $(z^{i^*})^\top z^{i^*}$  is required only once. Thus, the steepest edge method for the dual without the slack variables requires fewer inner product calculations than the steepest edge algorithm for the primal in the space of basic and nonbasic variables. In the 1994 annual Mathematical Programming Society meeting, Robert Bixby reported that running CPLEX on the NETLIB test set required about 22% more work per iteration for dual simplex with dual steepest edge pricing than dual simplex with partial pricing. The primal steepest edge pricing requires about 200% more work per iteration than partial pricing. The effect of using dual steepest edge pricing over the most infeasible variable can be dramatic. Bixby reported solving test problem *hc1amb* in 25785 iterations using dual steepest edge pricing, whereas this problem was not solved after 530,000 pivots using the most infeasible variable rule. In Section 6.3, on the dual simplex algorithm we discussed several reasons for using dual simplex instead of primal simplex. One reason was that it just might be faster. Steepest edge pricing is faster with dual simplex than primal simplex, assuming the “correct” dual problem is used.

## 6.7 OTHER COMPUTATIONAL ISSUES

### 6.7.1 Tolerances

Any practical implementation of the simplex algorithm is done in finite precision on a computer. This leads to round-off error problems during pivoting. Related to this is selecting a tolerance on zero. A tolerance on zero is needed in several places during the simplex algorithm.

1. A tolerance on row and column residuals. The stored value of  $\bar{x}_B$  is used to calculate  $A_B \bar{x}_B$ . If the row residual,  $A_B \bar{x}_B - b$  differs from 0 by more than a specified tolerance excessive rounding error is present and the basis should be inverted from scratch. Similarly for the column residual  $c_B - A_B^\top \bar{u}$ .
2. A feasibility tolerance  $\epsilon > 0$  on zero. Assume lower bounds of 0 and upper bounds of  $\infty$ . After reinverting the matrix, the solution should satisfy  $x_B \geq -\epsilon$ . Similarly, a variable  $j \in N$  is not considered for entry in the basis if  $\bar{w}_j \geq -\epsilon$ .

3. A pivot threshold tolerance  $\bar{a}_{ij}$ . In the minimum ratio test only consider those  $\bar{a}_{ij}$  which exceed the pivot size threshold. Going below this threshold may result in numerical difficulties. This is the topic of the next subsection.

To give a reader a feel for the order of magnitude for some of these numbers, in OSL the feasibility tolerance is  $10^{-8}$ . In LINDO this value is initially  $.8 \times 10^{-4}$  and is then tightened to  $10^{-5}$ . The LINDO tolerance on a zero pivot is  $10^{-10}$ .

### 6.7.2 Minimum Ratio Test

Problems arise when the pivot element  $\bar{a}_{i^*q}$  is too small. The problem of small pivot elements in inverting a matrix is well known and we address this topic further in Chapter 13. For now consider a modification of the ratio test developed earlier in order to avoid pivoting on elements that may be too small. Harris [223] proposed a perturbation method for the minimum ratio test in order to give more flexibility in selecting the pivot element. We refer to the minimum ratio test described in equation (5.11) in Chapter 5 as the *textbook minimum ratio test* since this is the way it is described in virtually all linear programming text books. The Harris variation on the textbook minimum ratio test uses an  $\epsilon > 0$ , a feasibility tolerance, and selects the exit variable in a two pass procedure. In the first pass calculate the ratios

$$\theta_P := \min\{(\bar{x}_{B_i} + \epsilon)/\bar{a}_{iq} \mid \bar{a}_{iq} > 0, i = 1, \dots, m\}. \quad (6.27)$$

Given  $\theta_P$ , determine the index of the exit variable by

$$i^* = \operatorname{argmax} \{\bar{a}_{iq} \mid (\bar{x}_{B_i}/\bar{a}_{iq}) \leq \theta_P, \bar{a}_{iq} > 0, i = 1, \dots, m\}. \quad (6.28)$$

The increased flexibility in selecting the pivot element in (6.28) adds to the numerical stability of the simplex algorithm.

Because the selected exit variable is not necessarily the variable with the minimum ratio, it is possible that a basic variable becomes negative. However, it remains larger than  $-\epsilon$  which is, in effect, its new lower bound. There are two potential problems if a basic variable with negative value is pivoted out of the basis. First, the step length is  $\alpha_P = \bar{x}_{B_{i^*}}/\bar{a}_{i^*q}$  (refer back to equation (5.12) in Chapter 5). If  $\bar{x}_{B_{i^*}} < 0$  this leads to negative step length and a decrease in the objective function value. Second, such a step may also cause basic variables in rows where  $\bar{a}_{iq} < 0$  to go negative. These two problems are overcome by defining the step length as

$$\alpha_P := \max \{(\bar{x}_{B_{i^*}}/\bar{a}_{i^*q}), 0\}. \quad (6.29)$$

Unfortunately, using a zero step length does not solve all of our troubles. In the simplex algorithm, the nonbasic variables are not "calculated." They are automatically set to their lower bound zero. In order to simplify our discussion we assume that all variables initially have a lower bound of zero and an upper bound of  $\infty$ . If the lower bound on a basic variable is zero, then pivoting it out of the basis results in that variable being reset to zero. This may result in an error larger than machine precision in satisfying the constraints  $Ax = b$ . An answer to the problem of a negative  $\bar{x}_{B,*}$  suggested by Gill et al. [183] is to actually modify the lower bound on basic variable  $x_{B,*}$  to  $\bar{x}_{B,*}$ . These bound shifts obviously result in a changed problem. After optimality is achieved, if any variables remain at their shifted lower bound, reenter a Phase I. The IBM OSL package actually works with a combination Phase I and Phase II objective function and will allow pivots which hurt the true objective function in order to improve the numerical stability of the problem. See Forrest and Tomlin [157]. It may be desirable to go slightly infeasible in order to improve numerical stability. See Forrest and Tomlin [157] and Wolfe [458].

**Example 6.13** This example is from Gill et al. [183]. Assume the current basis is,

$$\begin{bmatrix} \bar{x}_{B_1} \\ \bar{x}_{B_2} \end{bmatrix} = \begin{bmatrix} .0009 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} \bar{a}_{1q} \\ \bar{a}_{2q} \end{bmatrix} = \begin{bmatrix} .1 \\ 100 \end{bmatrix}$$

and variable  $x_q$  is selected to enter the basis. By the textbook ratio test row one is selected as the pivot row because

$$(\bar{x}_{B_1}/\bar{a}_{1q}) = (.0009/.1) = .009 < (\bar{x}_{B_2}/\bar{a}_{2q}) = (1/100) = .01.$$

This results in a pivot on the element 0.1. Now perform the Harris ratio test using  $\epsilon = 0.001$ .

$$\frac{\bar{x}_{B_1} + \epsilon}{\bar{a}_{1q}} = \frac{.0009 + .001}{.1} = .019, \quad \frac{\bar{x}_{B_2} + \epsilon}{\bar{a}_{2q}} = \frac{1 + .001}{100} = .01001$$

This gives  $\theta_P = .01001$ . Then

$$i^* = \operatorname{argmax} \{ \bar{a}_{iq} \mid (\bar{x}_{B_i}/\bar{a}_{iq}) < \theta_P = .01001 \} = 2$$

If  $i^* = 2$  then the pivot element is 100 and  $\alpha_P = (\bar{x}_{B_2}/\bar{a}_{2q}) = 0.01$  and the new solution is

$$\begin{bmatrix} \bar{x}_{B_1} \\ \bar{x}_{B_2} \end{bmatrix} = \begin{bmatrix} .0009 \\ 1 \end{bmatrix} - .01 \begin{bmatrix} .1 \\ 100 \end{bmatrix} = \begin{bmatrix} -.0001 \\ 0 \end{bmatrix}$$

The basic variable in row one is now negative, but is greater than  $\epsilon$ . Later, in Example 13.5 we continue with this example and show how the numerical accuracy of the problem is greatly improved by selecting the larger pivot element.

**Table 6.1** Sequence of Basic Solutions Using Textbook Minimum Ratio Test

<i>Iteration 1</i>	<i>Iteration 2</i>	<i>Iteration 3</i>	<i>Iteration 4</i>	<i>Iteration 5</i>
$x_3 = 1.0$	$x_3 = 0$	$x_2 = 0$	$x_2 = .003$	$x_2 = 3.999$
$x_4 = 1.1$	$x_1 = 1$	$x_1 = 1$	$x_1 = 1.002$	$x_1 = 3.0$
$x_5 = 2.001$	$x_5 = 0.001$	$x_5 = 0.001$	$x_4 = 0.008$	$x_4 = 1.799$
$x_6 = 3.0$	$x_6 = 2.0$	$x_6 = 2.0$	$x_6 = 1.998$	$x_3 = 0.666$

Even if we are working in infinite precision, and numerical stability was not an issue, the Harris modification of the ratio test can help overcome degeneracy and help at “congested corners.”

### Example 6.14

$$\begin{array}{ccccccccc} \min & -2x_1 & & & & & & & \\ & 1x_1 & -(1/1.5)x_2 & +x_3 & & & & = & 1 \\ & 1.1x_1 & & -x_2 & +x_4 & & & = & 1.1 \\ & 2x_1 & & -x_2 & & +x_5 & & = & 2.001 \\ & x_1 & & & & & +x_6 & = & 3 \end{array}$$

The initial basic solution is

$$x_3 = 1, \quad x_4 = 1.1, \quad x_5 = 2.001, \quad x_6 = 3.$$

The textbook ratio test gives the sequence of basic solutions listed in Table 6.1. (Break ties using the largest pivot element.) Now use the Harris ratio test for the first basis with  $\epsilon = 0.001$ .

$$\begin{aligned} \theta_P &= \min\left\{\frac{1+\epsilon}{1}, \frac{1.1+\epsilon}{1.1}, \frac{2.001+\epsilon}{2}, \frac{3+\epsilon}{1}\right\} \\ &= \min\left\{1 + \frac{\epsilon}{1}, 1 + \frac{\epsilon}{1.1}, 1 + \frac{.001+\epsilon}{2}, 3 + \epsilon\right\} \\ &= 1 + \frac{\epsilon}{1.1} = 1 + \frac{0.001}{1.1} = 1.00091 \end{aligned}$$

Since  $(2.001/2) = 1.0005 < 1.00091$ , the basic variables in rows one, two and three are the candidates for the exit variables in the second part of the ratio test which is

$$\begin{aligned} i^* &= \operatorname{argmax} \{ \bar{a}_{i1} \mid (\bar{x}_{B_i}/\bar{a}_{i1}) \leq \theta_P, \bar{a}_{i1} > 0, i = 1, \dots, m \} \\ &= \operatorname{argmax} \{ 1, 1.1, 2 \} = 3. \end{aligned}$$

Pivot in variable  $x_1$  and pivot out  $x_5$ . The initial tableau and the tableau after the pivot are

$$\begin{array}{rccccccccc} z_0 & & 2x_1 & & = & & 0 \\ x_3 & & +1x_1 & -(1/1.5)x_2 & = & & 1 \\ x_4 & & 1.1x_1 & -x_2 & = & & 1.1 \\ x_5 & & +2x_1 & -x_2 & = & & 2.001 \\ x_6 & & +1x_1 & & = & & 3 \end{array}$$

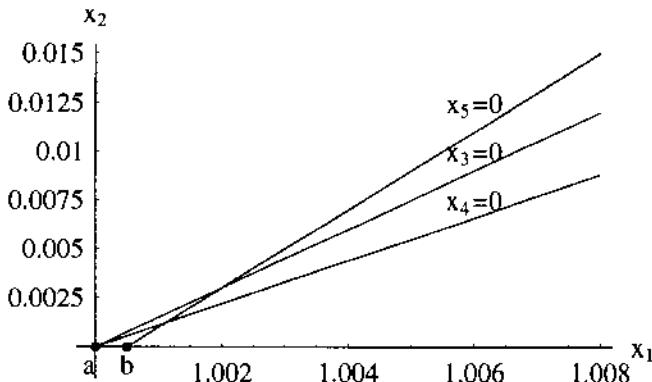
$$\begin{array}{rccccccccc} z_0 & & -x_5 & +x_2 & = & & -2.001 \\ x_3 & & -.5x_5 & -.25x_2 & = & & -.00050 \\ x_4 & & -.55x_5 & -.45x_2 & = & & -.00055 \\ x_1 & & +.5x_5 & -.5x_2 & = & & 1.0005 \\ x_6 & & -.5x_5 & +.5x_2 & = & & 1.9995 \end{array}$$

After pivoting in variable  $x_1$  we have two basic variables at a negative value, but the value is less than  $\epsilon$ . Now pivot in variable  $x_2$  and pivot out variable  $x_6$ . This gives the optimal basis. Only two iterations after the initial basis are required using the Harris ratio test versus four iterations using the textbook ratio test. What happened? This is not an issue of numerical stability. Rather, by allowing a variable to go slightly negative, we move past the degenerate extreme point corresponding to the basis  $\{x_1, x_2, x_5, x_6\}$  and then go feasible in one step. See Figure 6.2. Point  $a$  is a degenerate extreme point of the feasible region. It corresponds to the basic feasible solutions of Iteration 2 and Iteration 3 in Table 6.1. Point  $b$  corresponds to the basis  $\{x_3, x_4, x_1, x_6\}$  and  $x_3 \geq 0, x_4 \geq 0$  are slightly violated. At the next pivot, we move along the edge corresponding to  $x_5 = 0$  to the optimal extreme point.

### 6.7.3 Scaling

We have documented the problems caused by dividing by small pivot elements. This problem may be exacerbated if the constraint matrix contains elements of vastly differing magnitude. Although there is not a rigorous definition of a *well scaled* matrix, we say a matrix is well scaled if the elements are of similar magnitude. Some linear programming codes such as LINDO will give a poorly scaled warning message if the largest nonzero element in absolute value divided by the smallest nonzero element in absolute value exceeds a certain quantity such as  $10^6$ . The discussion in this subsection is taken from Tomlin [432].

A linear program is scaled by pre-multiplying the constraint matrix  $A$  by the diagonal matrix  $D_1$  and post-multiplying by the diagonal matrix  $D_2$ . If the

**Figure 6.2** Illustration of Harris Ratio Test

elements of  $A$  are  $a_{ij}$ , these elements are stored in binary form as a mantissa (or significand) and a power of 2 (or perhaps 16). See Appendix B. An important point made by Tomlin is that it is best not to alter the significance of the user's input data. Therefore, the diagonal elements of  $D_1$  and  $D_2$  should be powers of 2 in order to maintain the user's mantissa. Denote by  $2^{h_{1i}}$ , the scaling factor of diagonal element  $i$  of  $D_1$ . Similarly for  $2^{h_{2j}}$  for elements of  $D_2$ . If  $A' = D_1 A D_2$  and  $e_{ij}'$  is the exponent of  $a_{ij}$ , then

$$e_{ij}' = e_{ij} + h_{1i} + h_{2j}.$$

Next define

$$\begin{aligned} e_{\max} &:= \max \{e_{ij}' \mid i = 1, \dots, m, j = 1, \dots, n\} \\ e_{\min} &:= \min \{e_{ij}' \mid i = 1, \dots, m, j = 1, \dots, n\} \end{aligned}$$

and let  $\mu$  and  $\sigma^2$  represent the mean and variance, respectively, of the  $e_{ij}'$ . Proposed scaling methods include

1. *equilibration*: scale each row to make the largest row element 1.0 and then scale the columns to make the largest column element 1.0;
2. *geometric mean*: for each row calculate  $\sqrt{\max_j |a_{ij}| / \min_j |a_{ij}|}$  and then divide the row by the number to the power of 2 closest to this, then do the same for each column;
3. *arithmetic mean*: divide each row by the number to the power of 2 which is the closest to the row mean, similarly for the columns.

In addition to these simple methods, several optimization schemes have been proposed. Fulkerson and Wolfe [164] proposed a method based on a network flow labeling algorithm for finding the  $h_{1i}$  and  $h_{2j}$  such that  $e_{\max} - e_{\min}$  is minimized. Curtis and Reid [107] proposed a method for minimizing  $\sigma^2$ .

Tomlin reports the effect of these different scaling methods on several test problems. He reports the effect of scaling on sparsity, condition number of the bases, residual norms, relative error, etc. Unfortunately, based on his experiments it is difficult to come to any conclusions about the power of his scaling. There are two interesting aspects of Tomlin's results. First, scaling can actually increase the number of nonzero elements in the basis inverse. In many real applications there are very few distinct nonzero elements. In fact, many nonzero elements tend to be  $\pm 1$ . Scaling can destroy this structure and in turn cause fewer cancellation of nonzero elements during Gaussian elimination. Second, good formulation is far more important than a mechanized scaling routine. For example, it is important use consistent units of measure. Do not use grams as a unit of measure in one constraint and kilograms as a unit of measure in another. Tomlin suggests that one reason for the poor showing of scaling on his test problems is that they were all formulated by experienced modelers.

## 6.8 CONCLUSION

Sensitivity analysis is covered in most linear programming text books. See Bazaraa, Jarvis, and Sherali [43] for a very thorough coverage. See also the book by Greenberg [208]. The dual simplex algorithm is due to Lemke [283]. Bixby [54] gives a brief description of the dual simplex method used in CPLEX. Dual simplex is a viable alternative to primal simplex. Two good references for finding a starting basis and Phase I simplex are Bixby [54] and Greenberg [210]. Forrest and Goldfarb [153] were the first to show the practicality of the steepest edge pricing. Many modern implementations of the ratio test are based on the ideas in Gill et al. [183] and Harris [223]. Readers interested in further information on the numerical linear algebra aspects of the simplex algorithm should consult Chapter 5 of Murtagh [348] and Orchard-Hays [358]. Chapter 4 in the book by Duff and Reid [132] contains an excellent discussion of the problems caused by finite precision when solving a system of linear equations.

## 6.9 EXERCISES

- 6.1 Find the allowable increase and decrease on the right hand side of constraint  $(7/10)x_1 + x_2 \leq 630$  in the linear program

$$\begin{aligned} \max \quad & 10x_1 + 9x_2 \\ (7/10)x_1 + x_2 & \leq 630 \\ (1/2)x_1 + (5/6)x_2 & \leq 600 \\ x_1 + (2/3)x_2 & \leq 708 \\ (1/10)x_1 + (1/4)x_2 & \leq 135 \\ x_1, x_2 & \geq 0 \end{aligned}$$

using the information available from the optimal simplex tableau.

- 6.2 Find the allowable increase and decrease on the objective function coefficient of variable  $x_1$  in the linear program

$$\begin{aligned} \max \quad & 10x_1 + 9x_2 \\ (7/10)x_1 + x_2 & \leq 630 \\ (1/2)x_1 + (5/6)x_2 & \leq 600 \\ x_1 + (2/3)x_2 & \leq 708 \\ (1/10)x_1 + (1/4)x_2 & \leq 135 \\ x_1, x_2 & \geq 0 \end{aligned}$$

using the information available from the optimal simplex tableau.

- 6.3 Prove Proposition 6.1.

- 6.4 What is the binary representation of .1 using a 32 bit word (see Appendix B)?

- 6.5 Prove Proposition 6.11.

- 6.6 Modify the Harris ratio test to handle simple lower and upper bounds on the variables.

- 6.7 Develop a Phase I procedure for the dual simplex algorithm.

- 6.8 Construct a linear program which has a unique optimal primal solution, but the allowable increase of a right hand side calculated using the simplex algorithm (see equation (6.2)) strictly underestimates the true allowable increase of the constraint right hand side.

6.9 Continue with Example 6.10 until an optimal solution is found.

6.10 Consider the linear knapsack problem

$$\begin{aligned} \min \quad & \sum_{j=1}^n c_j x_j \\ \sum_{j=1}^n a_j x_j - x_{n+1} & = b \\ x_j & \geq 0, \quad j = 1, \dots, n+1. \end{aligned}$$

Assume  $c_j, a_j \geq 0$  for all  $j$ . Start with the primal infeasible solution  $x_{n+1} = -b$ ,  $\bar{x} = 0$ . Prove that dual simplex algorithm finds the optimal solution in one pivot.

6.11 Assume  $\bar{x}$  is a feasible solution for a minimization linear program. Show how to construct a basic feasible solution  $\hat{x}$  from  $\bar{x}$  such that  $c^\top \hat{x} \leq c^\top \bar{x}$ .

6.12 Assume partial pricing is used in the revised simplex algorithm. What difficulty does this cause for the dual update in Step 6 of Algorithm 5.9?

6.13 A *free variable* is a variable with a lower bound of  $-\infty$  and an upper bound of  $+\infty$ . If  $x$  is a free variable in a linear program, it can be replaced by the two variables  $x', x''$ , with lower bounds of 0 and upper bounds of  $\infty$  by making the substitution  $x = x' - x''$ . In any basic feasible solution, both  $x'$  and  $x''$  will not be positive. Why? Now explain how to modify the simplex algorithm when a subset of the variables are free variables, so the substitution method just described is not needed.

6.14 Apply the artificial basis Phase I procedure to the Example 5.2 and obtain a starting basic feasible solution.

6.15 Construct an example linear program that has an optimal solution and is both primal and dual degenerate.

6.16 Prove Proposition 6.5.

---

# INTERIOR POINT ALGORITHMS: POLYHEDRAL TRANSFORMATIONS

## 7.1 INTRODUCTION

The simplex algorithm is not a polynomial algorithm for solving the linear programming problem. The first polynomial algorithm for linear programming, the ellipsoid algorithm, is due to L. G. Khachiyan [262, 263]. This was met with great excitement from the mathematical programming community and stimulated considerable research. See Bland, Goldfarb, and Todd [63] for a good survey. There are also many important consequences of the ellipsoid algorithm in combinatorial optimization. See, for example, the book by Grötschel, Lovász, and Schrijver [212]. Unfortunately, the ellipsoid performed very poorly from a computational standpoint. Then in 1984, N. K. Karmarkar [260] created new excitement with claims of an algorithm that not only was polynomial in complexity, but also outperformed the simplex algorithm on large sparse problems. This created an incredible amount of research on *interior point* methods.

The simplex algorithm moves along the boundary of a polyhedron from extreme point to extreme point changing only one basic variable at each iteration. There are a finite number of extreme points so the simplex algorithm is a discrete or “combinatorial” algorithm. The philosophy of interior point algorithms is very “nonlinear.” Interior point algorithms move in the feasible direction which gives the maximum (per unit distance) improvement in the objective function value after projection into a linear subspace. This feasible direction will generally be through the interior of the polyhedron. Consider the linear programming problem in standard form

$$\begin{aligned} & \min c^\top x \\ (LP) \quad & \text{s.t. } Ax = b \end{aligned}$$

$$x \geq 0$$

Assume  $x^0 > 0$  is a feasible solution to this problem. The direction of steepest descent is given by the negative of the gradient of  $c^\top x$  at  $x^0$ . Since the objective function is linear, the direction of steepest descent is  $-c$ . Ideally, we want to move as far in this direction as possible. Unfortunately, this may not be a feasible direction because moving in this direction may violate the  $Ax = b$  constraints. Define a linear subspace of  $\mathbb{R}^n$  by  $H = \{x \in \mathbb{R}^n \mid Ax = 0\}$ . Since  $x^0 > 0$ , if  $\hat{c} \in H$ , then for sufficiently small  $\alpha > 0$ ,  $x^0 - \alpha\hat{c}$  is a feasible solution to the linear program (*LP*). Of course, we want  $-\hat{c}$  to be an improving direction, i.e. make the objective function smaller. For every  $u$ , the vector  $A^\top u$  is in the space orthogonal to  $H$  and there is a  $\bar{u}$  such that  $c$  may be written as the unique sum of the orthogonal vectors  $c = \hat{c} + A^\top \bar{u}$  where  $\hat{c} \in H$  and  $A^\top \bar{u} \in H^\perp$ . Thus,  $\hat{c}$  is the *projected gradient* of  $c$  onto  $H$ . See Section A.5 in Chapter A. It is left as an exercise to show that if  $A$  has rank  $m$  the *least squares* optimization problem  $\min_u \|c - A^\top u\|^2$  has the unique solution  $\bar{u} = (AA^\top)^{-1}Ac$ . If

$$\hat{c} = c - A^\top \bar{u} = c - A^\top (AA^\top)^{-1}Ac$$

then

$$A\hat{c} = Ac - AA^\top \bar{u} = Ac - AA^\top (AA^\top)^{-1}Ac = 0$$

which implies  $\hat{c} \in H$  is a feasible direction and there is an  $\alpha > 0$  such that  $x^0 - \alpha\hat{c}$  is a feasible solution to (*LP*). If  $\hat{c}$  is nonzero then  $-\hat{c}$  is an improving direction since

$$\begin{aligned} c^\top (x^0 - \alpha\hat{c}) &= c^\top x^0 - \alpha c^\top \hat{c} \\ &= c^\top x^0 - \alpha(\hat{c} + A^\top (AA^\top)^{-1}Ac)^\top \hat{c} \\ &= c^\top x^0 - \alpha(\hat{c})^\top \hat{c} = c^\top x^0 - \alpha\|\hat{c}\|^2 \end{aligned}$$

where the last equality follows because  $\hat{c}$  is orthogonal to  $A^\top \bar{u} = A^\top (AA^\top)^{-1}Ac$ . Then  $\alpha > 0$  implies  $c^\top x^0 < c^\top x^0 - \alpha\|\hat{c}\|^2$ .

**Example 7.1** Consider the following two variable linear program from Anderson, Sweeney, and Williams [12].

$$\begin{array}{llll} \text{min} & -10x_1 - 9x_2 \\ \text{s.t.} & 7x_1 + 10x_2 & \leq & 6300 \\ & 3x_1 + 5x_2 & \leq & 3600 \\ & 3x_1 + 2x_2 & \leq & 2124 \\ & 2x_1 + 5x_2 & \leq & 2700 \\ & x_1, x_2 & \geq & 0 \end{array}$$

Converting to standard form gives

$$\begin{array}{ll} \min & -10x_1 - 9x_2 \\ \text{s.t.} & 7x_1 + 10x_2 + x_3 = 6300 \\ & 3x_1 + 5x_2 + x_4 = 3600 \\ & 3x_1 + 2x_2 + x_5 = 2124 \\ & 2x_1 + 5x_2 + x_6 = 2700 \\ & x_1, x_2 \geq 0 \end{array}$$

An initial feasible point is  $x^0 = (1, 0.5, 6288, 3594.5, 2120, 2695.5)$  with objective function value -14.5. The gradient of the objective function is  $c^\top = [10 \ 9 \ 0 \ 0 \ 0 \ 0]^\top$ . The projected gradient,  $\hat{c}$ , on the null space  $\{x \mid Ax = 0\}$  is

$$\hat{c} = (I - A^\top (AA^\top)^{-1} A) c = \begin{bmatrix} -0.668405 \\ 0.377477 \\ 0.904067 \\ 0.117831 \\ 1.25026 \\ -0.550574 \end{bmatrix}$$

Now move  $\alpha$  units in the direction  $-\hat{c}/\|\hat{c}\|$ . The maximum possible value for  $\alpha$  is

$$\alpha = \min_i \{x_i^0 / (\hat{c}_i / \|\hat{c}\|) \mid \hat{c}_i > 0\}.$$

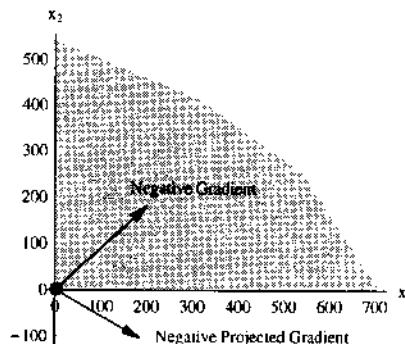
The new point is

$$x^1 = x^0 - \alpha \hat{c} / \|\hat{c}\| = \begin{bmatrix} 1 \\ 0.5 \\ 6288 \\ 3594.5 \\ 2120 \\ 2695.5 \end{bmatrix} - 2.4014 \begin{bmatrix} -0.368685 \\ 0.208212 \\ 0.498674 \\ 0.0649944 \\ 0.689631 \\ -0.303691 \end{bmatrix} = \begin{bmatrix} 1.88536 \\ 0 \\ 6286.8 \\ 3594.34 \\ 2118.34 \\ 2696.23 \end{bmatrix}$$

The objective value improves from -14.5 to -18.8536.

The algorithm just described and illustrated in Example 7.1 is the *gradient projection algorithm* and is due to Rosen [387]. There are two drawbacks to this algorithm. First, the projected gradient  $-\hat{c}$ , may differ significantly from the direction of steepest descent  $-c$ . Second, if the starting point  $x^0$  is near the boundary of the linear programming polyhedron then it may not be possible to move very far in the direction  $-\hat{c}$  without hitting the boundary. These problems are illustrated in Example 7.1 and the corresponding Figure 7.1. In this Figure the polyhedron has been projected onto  $(x_1, x_2)$  space along with

**Figure 7.1** Polytope Projected Onto  $x_1, x_2$  Space



the negative gradient and the projected negative gradient. The initial point  $x^0$  with component  $x_2^0 = 0.5$  is very close to the boundary facet  $x_2 \geq 0$ . This is a problem because the corresponding component of the projected gradient is  $\hat{c}_2 = 0.377377$  and only a small move of 2.4014 units in the direction  $-\hat{c}/\|\hat{c}\|$  is possible. Notice also that  $c$  and  $\hat{c}$  differ significantly. In fact, the angle between  $c$  and  $\hat{c}$  is approximately 82 degrees. To better understand this issue, recall that with the simplex algorithm one pivot strategy was to pivot on the variable with the largest negative reduced cost. Unfortunately, this turns out to be a poor strategy if this variable is pivoted in at a very small value. A variable that looks poor in terms of reduced cost may be a good variable to pivot in if it comes in at a large value resulting in a big reduction in the objective function value. The *key idea* behind all interior point algorithms is to avoid being too close to the boundary of the polytope so that moving in the direction of the negative projected gradient results in significant improvement.

In this chapter we describe interior point algorithms which are based on polyhedral transformations. At each iteration of these algorithms, the polyhedron is transformed so that the current iterate is “centered” and it is possible to take long steps in the direction of the negative projected gradient. An alternative philosophy based on barrier functions is described in Chapter 8. Before developing any of the interior point algorithms in detail we motivate the polyhedral transformation idea through an example.

**Example 7.2 (Example 7.1 continued.)** Use the initial solution  $x^0$  to define the following matrix.

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6288 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3594.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2120 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2695.5 \end{bmatrix}$$

Transform the linear program of Example 7.1 by  $x = Xy$ . The new linear program is

$$\begin{aligned} \min \quad & -10y_1 - 4.5y_2 \\ \text{s.t.} \quad & 7y_1 + 5y_2 + 6288y_3 = 6300 \\ & 3y_1 + 2.5y_2 + 3594.5y_4 = 3600 \\ & 3y_1 + y_2 + 2120y_5 = 2124 \\ & 2y_1 + 2.5y_2 + 2695.5y_6 = 2693 \\ & y_1, y_2, y_3, y_4, y_5, y_6 \geq 0 \end{aligned}$$

The theoretical development of this transformation is given later in the chapter in Section 7.6. The important thing is to observe the structure of the transformed polytope in Figure 7.2 below. It has been "stretched" in the  $y_2$  direction. The transformation "pushes" the original initial point of  $(1, 0.5, 6288, 3594.5, 2120, 2695.5)$  away from the  $y_2 \geq 0$  facet to the new initial point  $y^0 = X^{-1}x^0 = (1, 1, 1, 1, 1, 1)$ . There is now a distance of at least one unit to the facets  $y_1 \geq 0$  and  $y_2 \geq 0$ . The transformed initial point is "centered" at least 1 unit from each of the nonnegativity facets.

The transformed objective function is  $(c^T X)y$  and its gradient is  $c^T X = [-10 - 4.5 0 0 0 0]^T$  and its projection into the linear space  $AXy = 0$  is (after normalization)  $[-9.99994 - 4.49996 0.0147105 0.0114758 0.0162735 0.0115933]^T$ . The angle between the gradient and projected gradient is reduced to .14 degrees from 82 degrees. It is possible now to move 673 units in the direction of the normalized negative projected gradient. The new point is

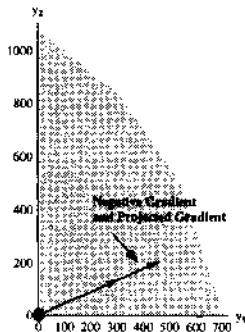
$$y^1 = (615.493, 277.521, 0.096047, 0.294817, 0, 0.287595).$$

Then

$$x^1 = Xy^1 = (615.493, 138.761, 603.944, 1059.72, 0, 775.211).$$

The new objective function value at  $x^1$  is -7403.77, a considerable improvement over -18.8536. The transformation moves the initial point near the center of

**Figure 7.2** Transformed Polytope Projected Onto  $y_1, y_2$  Space



*the polytope which allows for moving more closely in the direction of steepest descent, thus allowing for a bigger step size.*

In the next section we develop the basic method of Karmarkar for transforming polytopes. This transformation method is used in Section 7.3 where Karmarkar's original polynomial linear programming algorithm is given. The proof that the algorithm is polynomial is given in Section 7.4. Since Karmarkar's algorithm is an interior point algorithm, it does not terminate with the optimal extreme point. Finding an optimal extreme point using Karmarkar's algorithm is detailed in Section 7.5. How a linear program is converted into the format required by Karmarkar is also discussed in Section 7.5. The affine scaling algorithms which are alternatives to the Karmarkar algorithm are developed in Section 7.6. In Section 7.7 we provide a discussion which shows how the methods of this chapter are really weighted least squares algorithms. Concluding remarks are in Section 7.8. Exercises are provided in Section 7.9.

In Chapter 8 we develop barrier methods as alternatives to polyhedral transformation. The barrier algorithms are the algorithms that are currently used with the greatest effectiveness in solving large scale linear programs. At this point the reader can go directly to the chapter on barrier methods without loss of continuity. However, the material in this chapter is still very important. Many algorithms for linear programming are based on the affine scaling methods and Karmarkar-like potential functions which are developed in this chapter.

## 7.2 PROJECTIVE TRANSFORMATIONS

In 1984 Karmarkar [260] proposed a method for solving linear programming problems which is similar to the algorithm of Rosen in that the projected gradient is used to calculate the direction of improvement and that moves across the interior of the feasible region are permitted. However, Karmarkar avoids the problem of being “too close to the boundary” (and therefore not being able to take a large step in direction of the negative projected gradient) by using a very clever transformation at each iteration. First, Karmarkar takes the original linear program and transforms it into a linear program whose polytope is fairly symmetrical in the sense that the ratio of the radius  $R$  of the smallest circumscribing sphere to the radius  $r$ , of the largest inscribed sphere is  $n - 1$ . This means that the polytope is not too long and skinny in any direction. Next, the current iterate  $x^k$ , is transformed to  $y^0$  which is the center of sphere in a problem which *approximates* the transformed linear program. The problem approximation is optimized in one step with  $\bar{y}$  being the optimal solution. The optimal approximate solution  $\bar{y}$  is then transformed back to the original space generating a new iterate  $x^{k+1}$  and the process is repeated. The problem which approximates the transformed optimization problem has the following structure.

$$\min f^\top y \quad (7.1)$$

$$\text{s.t. } Ky = 0 \quad (7.2)$$

$$y \in W_y \quad (7.3)$$

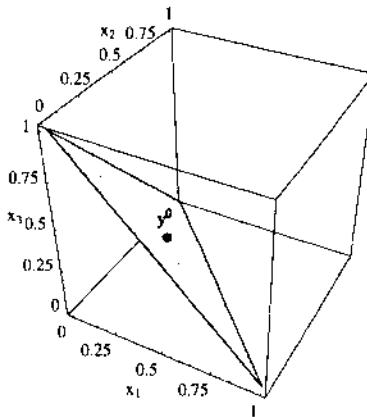
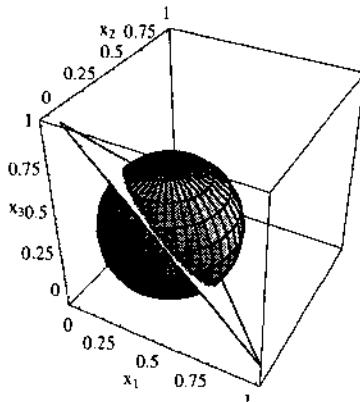
$$(y - y^0)^\top (y - y^0) \leq \alpha^2 r^2 \quad (7.4)$$

In this formulation  $f$  is an  $n$  component column vector,  $K$  an  $m \times n$  matrix of rank  $m$  and  $W_y$  is an  $n - 1$  dimensional *simplex* defined by

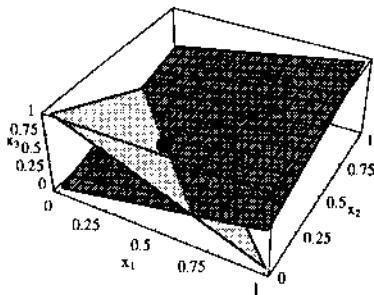
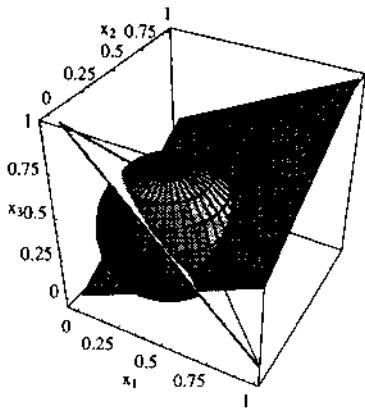
$$W_y := \{y \mid \sum_{i=1}^n y_i = 1, y_i \geq 0\}. \quad (7.5)$$

A two dimensional simplex in  $\mathbb{R}^3$  is illustrated in Figure 7.3.

The constraint (7.4) defines an  $n$  dimensional sphere with center  $y^0$  and radius  $\alpha r$  where  $0 < \alpha < 1$  and  $r = 1/\sqrt{n(n-1)}$ . Furthermore, a transformation  $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is constructed shortly with the property that  $T : x^k \rightarrow y^0 = (1/n, \dots, 1/n)$  and  $Ky^0 = 0$ . The geometry of this problem is very interesting. Because the sphere is centered at  $y^0 = (1/n, \dots, 1/n)$  and has radius  $\alpha r$ , with  $0 < \alpha < 1$ , the intersection of the sphere with the simplex  $W_y$  is an  $n - 1$  dimensional sphere with center at  $y^0$  and is completely contained in the interior

**Figure 7.3** A Two Dimensional Simplex in  $\mathbb{R}^3$ **Figure 7.4** Simplex and Sphere in  $\mathbb{R}^3$ 

of the simplex. It is left as an exercise to show that the maximum radius of an  $(n-1)$  sphere with center at  $(1/n, \dots, 1/n)$  that can be inscribed in the simplex  $S = \{(y_1, \dots, y_n) \mid \sum_{i=1}^n y_i = 1, y_i \geq 0, i = 1, \dots, n\}$  is  $r = 1/\sqrt{n(n-1)}$ . This is illustrated in Figure 7.4. Because  $Ky^0 = 0$  the linear space  $\{y \mid Ky = 0\}$  passes through the center of this  $n-1$  sphere. The rank of  $K$  is  $m$  so the feasible region defined by the constraints (7.2) - (7.3) is an  $n-m-1$  dimensional hyperplane through  $y^0$ . This is illustrated in Figure 7.5. Then the feasible region

**Figure 7.5** An  $n - m - 1$  Dimensional Hyperplane Through  $y^0$ .**Figure 7.6** The Feasible Region of the Transformed Polytope

given by (7.2) - (7.4) is a sphere intersecting this hyperplane and is therefore an  $n - m - 1$  dimensional sphere with center at  $y^0$ . This is illustrated in Figure 7.6.

This problem is optimized by gradient projection in one step! It is left as an exercise to show that if point  $y^0$  is the center of a sphere and there is a linear objective function, then the optimal solution  $\bar{y}$  is found by starting at  $y^0$  and stepping directly to the boundary of the sphere by moving in the direction of the negative gradient. In the problem considered here, it is a bit more complicated because of the constraints (7.2) and (7.3). However, the projection of  $f$  onto

the linear space  $\{y \mid Ky = 0, \sum_{i=1}^n y_i = 0\}$  is

$$\hat{f} = [I - P^\top (PP^\top)^{-1}P] f \quad (7.6)$$

where

$$P := \begin{bmatrix} K \\ e^\top \end{bmatrix}$$

and  $e^\top$  is a  $n$  dimensional row vector and every element is 1.0.

**Proposition 7.3** *The optimal solution to problem (7.1)-(7.4) is*

$$\bar{y} = y^0 - \alpha \tau \frac{\hat{f}}{\|\hat{f}\|}.$$

The major work involved in Karmarkar's algorithm requires the calculation of the inverse matrix  $(PP^\top)^{-1}$  in (7.6). This is done by finding the *Cholesky factorization* of  $(PP^\top)$  which is the *LU* decomposition of a symmetric matrix. Finding this projection is at the heart of all interior point algorithms. In Chapter 13 we discuss further methods for implementing the projection calculation which take advantage of the sparsity of  $P$ .

The problem defined by (7.1)-(7.4) is optimized in one step. How does one go from the original linear program to a problem with the special structure just studied? The canonical form of the linear program used by Karmarkar is

$$\min c^\top x \quad (7.7)$$

$$(KLP) \quad \text{s.t. } Ax = 0 \quad (7.8)$$

$$e^\top x = 1 \quad (7.9)$$

$$x \geq 0 \quad (7.10)$$

where  $c$  is an  $n$  dimensional column vector,  $e$  an  $n$  dimensional column vector defined earlier and  $A$  is an  $m \times n$  matrix with rank  $m$ . Constraints (7.9)-(7.10) define the simplex  $W_x = \{x \in \mathbb{R}^n \mid \sum_{i=1}^n x_i = 1, x_i \geq 0, i = 1, \dots, n\}$ . Two assumptions are made about the linear program (KLP) in standard form.

1. The point  $x^0 = (1/n, \dots, 1/n)$  is a feasible solution to the linear program (KLP).
2. The optimal solution value of the linear program (KLP) is zero.

At each iteration of the Karmarkar algorithm the linear program (*KLP*) is transformed to a new problem such that a restriction of the new problem is optimized in one step. We require the transformed problem to have the structure given by (7.1)-(7.3). Therefore, the transformation  $T_k(x) \rightarrow y$  should: 1) take the current iterate and scale it so that each component of  $y^k = T_k(x^k)$  is equal; and 2) project the point  $y^k$  onto a simplex  $W_y$  so that its components sum to one. To accomplish this, define a *projective transformation*  $T_k : \mathbb{R}^n \rightarrow \mathbb{R}^n$  by

$$y = T_k(x) = \frac{X_k^{-1}x}{e^\top X_k^{-1}x} \quad (7.11)$$

where  $X_k$  is the diagonal matrix

$$X_k = \begin{bmatrix} x_1^k & 0 & \cdots & 0 \\ 0 & x_2^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n^k \end{bmatrix}.$$

That is, the transformation is

$$y_i = \frac{(x_i/x_i^k)}{\sum_{j=1}^n x_j/x_j^k}. \quad (7.12)$$

It is necessary for  $x_i^k > 0$  for  $i = 1, \dots, n$  in order that the transformation be well defined. This transformation has three important properties.

**Proposition 7.4** *Under the transformation  $y = T_k(x)$  defined by (7.11):*

1. if  $x \in W_x$  and  $y = T_k(x) = \frac{X_k^{-1}x}{e^\top X_k^{-1}x}$ , then  $x = T_k^{-1}(y) = \frac{X_k y}{e^\top X_k y}$ ;
2. the image of  $x^k$  under  $T_k$  is  $y^k = T_k(x^k) = (\frac{1}{n}, \dots, \frac{1}{n})$ ;
3. the image of the simplex  $W_x = \{x \mid \sum_{i=1}^n x_i = 1, x_i \geq 0\}$  under  $T_k$  is the simplex  $W_y = \{y \mid \sum_{i=1}^n y_i = 1, y_i \geq 0\}$ .

**Proof:** We prove the first property. First show that  $(e^\top X_k y)(e^\top X_k^{-1}x) = 1$ . By definition of  $X_k$ ,  $y_i = (x_i/x_i^k)/(\sum_{j=1}^n x_j/x_j^k)$ . Then

$$(e^\top X_k y)(e^\top X_k^{-1}x) = \left( \sum_{i=1}^n x_i^k y_i \right) \left( \sum_{j=1}^n x_j/x_j^k \right)$$

$$\begin{aligned}
&= \left( \sum_{i=1}^n x_i / \left( \sum_{j=1}^n x_j / x_j^k \right) \right) \left( \sum_{j=1}^n x_j / x_j^k \right) \\
&= \sum_{i=1}^n x_i = 1.
\end{aligned}$$

But  $y = X_k^{-1}x/e^\top X_k^{-1}x$ , so  $x = (e^\top X_k^{-1}x)(X_k y)$ . But  $(e^\top X_k y)(e^\top X_k^{-1}x) = 1$ , so  $(e^\top X_k^{-1}x) = 1/(e^\top X_k y)$  and we have  $x = X_k y/e^\top X_k y$ .  $\square$

Using the transformation  $x = X_k y/e^\top X_k y$  the linear program (*KLP*) is transformed into

$$\begin{aligned}
&\min \frac{c^\top X_k y}{e^\top X_k y} \\
\text{s.t. } &(AX_k y)/(e^\top X_k y) = 0 \\
&y \in W_y
\end{aligned}$$

which is equivalent to (*KLP*) by Proposition 7.4. By assumption 2 the optimal objective function value is 0. Since  $y \in W_y$ ,  $e^\top X_k y$  is strictly positive and this optimization problem is equivalent to

$$\min c^\top X_k y \quad (7.13)$$

$$\text{s.t. } AX_k y = 0 \quad (7.14)$$

$$y \in W_y. \quad (7.15)$$

Rather than solving this problem, the following restriction is solved.

$$\min (f^k)^\top y \quad (7.16)$$

$$\text{s.t. } AX_k y = 0 \quad (7.17)$$

$$y \in W_y \quad (7.18)$$

$$(y - y^0)^\top (y - y^0) \leq \alpha^2 r^2 \quad (7.19)$$

where  $f^k = X_k c$ ,  $\alpha \in (0, 1)$ ,  $r = 1/\sqrt{n(n-1)}$  and  $y_0 = X_k^{-1}x^k/e^\top X_k^{-1}x^k = (1/n, \dots, 1/n)$ . This is identical to problem (7.1)-(7.4) and by Proposition 7.3, the optimal solution to this problem is  $\bar{y} = y^0 - r\alpha f^k / \|f^k\|$ . By part 1 of Proposition 7.4 we can transform this point back to the original space by  $x^{k+1} = X_k \bar{y}/e^\top X_k \bar{y}$ . Unfortunately,  $x^{k+1}$  is no longer the center of the simplex  $W_x$ . However, the *key insight* by Karmarkar is to transform  $x^{k+1}$  back into  $y$  space by the new transformation  $T_{k+1}$  and obtain a new problem (7.13)-(7.15). Because  $\alpha \in (0, 1)$  the sphere defined by  $(y - y^0)^\top (y - y^0) \leq \alpha^2 r^2$  never hits the boundary of the simplex  $W_y$ . Therefore, all the components of  $\bar{y}$  are strictly

positive. This implies that when  $\bar{y}$  is transformed back into  $x$  space under  $x^{k+1} = X_k \bar{y} / e^\top X_k \bar{y}$ , all the components of  $x^{k+1}$  are strictly positive. Then the matrix  $X_{k+1}$  has strictly positive diagonal elements and  $T_{k+1}$  is well defined. Then by Proposition 7.4,  $x^{k+1}$  is transformed back into the center of the simplex in  $y$  space. At every iteration we have a point that is at the center of the simplex in the transformed space and can take a “big step” in the direction of the negative projected gradient.

### 7.3 KARMARKAR'S ALGORITHM

The algorithm needs a stopping criterion. Assume without loss that the data in the formulation ( $KLP$ ) are integer. At this point the reader may wish to refer to Appendix B which introduces the concept of the size of a linear program. The size,  $L$  of a linear program with structure ( $KLP$ ) is

$$L = \sum_{i=1}^m \sum_{j=1}^n [1 + \log_2(1 + |a_{ij}|)] + (2n + m + 2) + \sum_{j=1}^n [1 + \log_2(1 + |c_j|)]. \quad (7.20)$$

In the definition of  $L$ , the term  $\sum_{i=1}^m \sum_{j=1}^n [1 + \log_2(1 + |a_{ij}|)] + (2n + m + 2)$  is the number of bits required to encode the constraints of the Karmarkar standard form linear program ( $KLP$ ) taking into account that  $b = 0$  and that the constraint  $e^\top x = 1$  requires  $2n + 2$  bits to encode. See equation B.2. This value of  $L$  is crucial later in Proposition 7.13 when we prove that an optimal extreme point solution with objective function value of zero can be found when Karmarkar's algorithm terminates. Actually, to prove Proposition 7.13 we could replace the binary encoding of the constraint matrix with the logarithm of the largest determinant in the linear program. However, one does not know this value a priori, so the value of  $L$  given in (7.20) is a conservative value.

#### Algorithm 7.5 (Karmarkar's Projective Algorithm)

**Step 1: (Initialization)**  $k \leftarrow 0$ ,  $x^0 \leftarrow (\frac{1}{n}, \dots, \frac{1}{n})$ ,  $r \leftarrow 1/n$ ,  $\alpha \in (0, 1)$ . Later we show polynomial termination for an  $\alpha = 1/3$ .

**Step 2: (Project)** Find the direction in which to step. This is the projection of  $f^k = X_k c$  onto the linear space  $P_k y = 0$  where

$$P_k := \begin{bmatrix} AX_k \\ e^\top \end{bmatrix}$$

$$\hat{f}^k \leftarrow (I - P_k^\top (P_k P_k^\top)^{-1} P_k)(f^k).$$

**Step 3: (Obtain new point)** *Move as far as possible in direction  $-\hat{f}^k$ .*

$$\bar{y} \leftarrow y^0 - \alpha r \frac{\hat{f}^k}{\|\hat{f}^k\|}$$

**Step 4: (Transform the variables)**

$$x^{k+1} \leftarrow \frac{(X_k \bar{y})}{(e^\top X_k \bar{y})}$$

**Step 5: (Termination Test)** *If  $c^\top x^k \leq 2^{-L} c^\top x^0$  stop; otherwise, increment  $k \leftarrow k + 1$  and return to Step 2.*

The values of  $1/3$  for  $\alpha$  and  $1/n$  for  $r$  are used later to simplify the proof of polynomial termination.

### Example 7.6

$$\begin{aligned} & \min x_1 + x_2 \\ \text{s.t. } & 3x_1 - x_2 + 4x_3 - 5x_4 = 0 \\ & x_1 + x_2 + x_3 + x_4 = 1 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

The optimal solution to this linear program is  $(0, 0, 5/9, 4/9)$ . The Mathematica implementation follows. Mathematica is product of Wolfram Research. Note how rapidly the objective function value is converging to 0.

```
(* MATHEMATICA IMPLEMENTATION OF KARMMARKAR PROJECTIVE
SCALING ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
alpha = .9;
n = 4;
m = 1;
radius= 1./n;
Iterlim = 10;
y0={{1./n}, {1./n}, {1./n}, {1./n}};
```

```

c={{1}, {1}, {0}, {0}};
A={{3, -2, 4, -5}};
e={{1.0}, {1.0}, {1.0}, {1.0}};
x0={{1./n}, {1./n}, {1./n}, {1./n}};
X = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = n;
For[j = 1, j <= n, j++,
  L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
  For[i = 1, i <= m, i++,
    L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] ];
];
L = L + 2*n + m + 2;
(* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-L)*Part[Transpose[c].x0, 1, 1];
(* NOW ITERATE ITERLIM TIMES *)
objval = Part[Transpose[c].x0, 1, 1];
iterkount = 1;
x = x0;
While[(iterkount <= Iterlim && objval > epsilon),
For[j = 1, j <= n, j++,
  X[[j,j]] = x[[j, 1]];
P = Append[A.X, {1., 1., 1., 1.} ];
f = X.c;
(* CALCULATE THE PROJECTED GRADIENT
OF THE OBJECTIVE FUNCTION *)
fhat = (IdentityMatrix[n] -
Transpose[P]. Inverse[(P.Transpose[P]).P]).f;
(* MOVE IN DIRECTION fhat *)
Norm = Sqrt[ Part[(Transpose[fhat].fhat), 1, 1]];
ybar = y0 - radius*alpha*fhat / Norm ;
(* NOW TRANSFORM BACK TO X SPACE *)
x = (X.ybar ) / Part[ ((Transpose[e].X).ybar), 1, 1];
objval = Part[Transpose[c].x, 1, 1];
Print["x = ", x];
Print["Objective Function Value = ", objval];
iterkount = iterkount + 1;
] (* END THE FOR STATEMENT *)

```

Here are the solution vectors.

```

x = {{0.149061}, {0.128032}, {0.38038}, {0.342527}}
Objective Function Value = 0.277093
x = {{0.0620654}, {0.0648845}, {0.478758}, {0.394292}}
Objective Function Value = 0.12695
x = {{0.0265284}, {0.0257037}, {0.523407}, {0.424361}}

```

```

Objective Function Value = 0.052232
x = {{0.0101336}, {0.0103316}, {0.543104}, {0.436431}}
Objective Function Value = 0.0204652
x = {{0.00395491}, {0.0039062}, {0.550738}, {0.441401}}
Objective Function Value = 0.0078611
x = {{0.00149219}, {0.00150398}, {0.553728}, {0.443276}}
Objective Function Value = 0.00299617
x = {{0.000570707}, {0.000567854}, {0.554859}, {0.444002}}
Objective Function Value = 0.00113856
x = {{0.00021574}, {0.000216429}, {0.555292}, {0.444276}}
Objective Function Value = 0.000432168
x = {{0.0000820677}, {0.0000819014}, {0.555455}, {0.444381}}
Objective Function Value = 0.000163969
x = {{0.0000310806}, {0.0000311208}, {0.555518}, {0.44442}}
Objective Function Value = 0.0000622014

```

## 7.4 POLYNOMIAL TERMINATION

In this section we show that Karmarkar's algorithm terminates in  $O(nL)$  steps. This proof of termination is adopted from Fang and Puthenpura [146].

**Motivation:** Karmarkar's algorithm terminates when  $c^\top x^k \leq 2^{-L} c^\top x^0$ . Following Karmarkar [260] we show that at each iteration  $k$ ,

$$c^\top x^k \leq e^{-k/5n} (c^\top x^0) \quad \text{for } k = 1, 2, \dots \quad (7.21)$$

In order to terminate it is then necessary to find a  $k$  such that  $e^{-k/5n} \leq 2^{-L}$ . That is

$$-k/5n \leq \ln(2^{-L}) \quad \text{or} \quad k \geq 5nL \ln(2) \quad (7.22)$$

But  $\ln(2) < 1$ , so the algorithm will terminate when  $k \geq 5nL$ . If (7.21) is shown then the number of steps in the algorithm is  $5nL$ . Taking the natural logarithm of (7.21) gives

$$\ln(c^\top x^k) \leq -k/5n + \ln(c^\top x^0), \quad (7.23)$$

$$n \ln(c^\top x^k) \leq -k/5 + n \ln(c^\top x^0). \quad (7.24)$$

Therefore, if  $c^\top x^k \leq e^{-k/5n} (c^\top x^0)$  for  $k = 1, 2, \dots$  is valid, the upper bound on  $n \ln(c^\top x^k)$  is reduced by  $1/5$  at every iteration. Unfortunately, the optimization problem solved at iteration  $k$  is

$$\min(f^k)^\top y \quad (7.25)$$

$$AX_k y = 0 \quad (7.26)$$

$$y \in W_y \quad (7.27)$$

$$(y - y^0)^\top (y - y^0) \leq \alpha^2 r^2 \quad (7.28)$$

with objective function  $(f^k)^\top y = c^\top X_k y$ , not  $n \ln(c^\top x^k)$ . However, the following Lemma guarantees a fixed decrease in the natural logarithm of the objective function,  $n \ln((f^k)^\top y)$ , when moving from the starting point  $y^0$  to the optimal point  $\bar{y}$ .

**Lemma 7.7** *If  $\bar{y} = y^0 - (\alpha r)(\hat{f}^k / \|\hat{f}^k\|)$  where  $\hat{f}^k = [I - P_k^\top (P_k P_k^\top)^{-1} P_k] f^k$ ,  $\alpha \in (0, 1)$ , and  $r = 1/n < 1/\sqrt{n(n-1)}$  then*

$$n \ln((f^k)^\top \bar{y}) \leq n \ln((f^k)^\top y^0) - \alpha. \quad (7.29)$$

**Proof:** Consider the following modification of the problem (7.25)-(7.28).

$$\min (f^k)^\top y \quad (7.30)$$

$$AX_k y = 0 \quad (7.31)$$

$$\sum_{j=1}^n y_j = 1 \quad (7.32)$$

$$(y - y^0)^\top (y - y^0) \leq R^2 = (n-1)/n \quad (7.33)$$

There are two differences between this problem and the problem defined by (7.25)-(7.28). First, the requirement that  $y \in W_y$  has been relaxed and we only require that  $\sum_{j=1}^n y_j = 1$ , that is, the nonnegativity constraints are deleted.

Second, in equation (7.33),  $R = \sqrt{\frac{n-1}{n}} > r$ . This is the radius of the smallest sphere that circumscribes the simplex  $W_y$ . The feasible region for problem (7.25)-(7.28) is a proper subset of the feasible region for problem (7.30)-(7.33). Refer back to Figure 7.6 for the feasible region of (7.25)-(7.28). A feasible region in  $R^3$  is illustrated in Figure 7.7. It is the intersection of two hyperplanes through the center of the sphere.

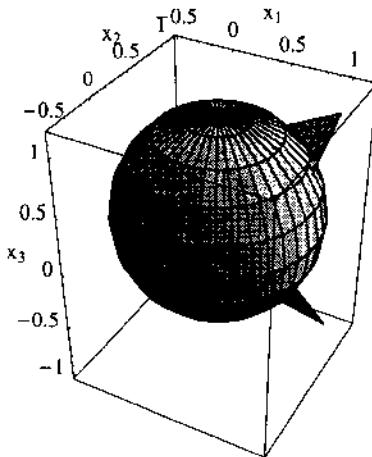
Proposition 7.3 applies to (7.30)-(7.33) and the optimal solution to this problem is  $\tilde{y} = y^0 - R(\hat{f}^k / \|\hat{f}^k\|)$  where as before,  $\hat{f}^k$  is the projection of  $f^k$  onto the linear space defined by  $P_k y = 0$ . Then  $(f^k)^\top \tilde{y} = (f^k)^\top y^0 - R \|\hat{f}^k\|$ , since  $(f^k)^\top \hat{f}^k = \|\hat{f}^k\|^2$ . Recall the following problem

$$\min (f^k)^\top y \quad (7.34)$$

$$AX_k y = 0 \quad (7.35)$$

$$y \in W_y \quad (7.36)$$

**Figure 7.7** A Relaxation of the Transformed Problem



which resulted from transforming the Karmarkar linear program (*KLP*) by  $x = X_k y / e^\top X_k y$ . Because the sphere defined in (7.33) circumscribes the simplex  $W_y$  and because the nonnegativity constraints which are included in (7.36) are deleted in (7.32), the formulation in (7.30)-(7.33) is a relaxation of the formulation (7.34)-(7.36). By assumption 2, the optimal solution value of (*KLP*) is zero which implies the optimal solution value of (7.34)-(7.36) is zero. Since (7.30)-(7.33) is a relaxation of this problem it has nonpositive objective function value which implies  $(f^k)^\top \bar{y} = (f^k)^\top y^0 - R\|\hat{f}^k\| \leq 0$  so  $\|\hat{f}^k\| \geq f^k y^0 / R$ . Since  $R = \sqrt{(n-1)/n} < 1$ , it follows that

$$(f^k)^\top \bar{y} = (f^k)^\top y^0 - \frac{\alpha}{n} \|\hat{f}^k\| \leq (1 - \frac{\alpha}{nR})(f^k)^\top y^0 < (1 - \frac{\alpha}{n})(f^k)^\top y^0.$$

Then  $\ln((f^k)^\top \bar{y}) \leq \ln((1 - \alpha/n)(f^k)^\top y^0) = \ln((f^k)^\top y^0) + \ln(1 - \alpha/n)$  and the result now follows from the fact that  $\ln(1 - \alpha/n) \leq -\alpha/n$ .  $\square$

Lemma 7.7 is true even if the optimal solution value of (*KLP*) is negative. This is significant later when putting a linear program into the Karmarkar standard form and the optimal objective function value is not known.

Unfortunately, a decrease in  $n \ln((f^k)^\top \bar{y})$  at each iteration does not imply a decrease in  $n \ln(e^\top x^k)$  at each iteration. Karmarkar overcomes this difficulty by using a function closely related to  $n \ln(e^\top x^k)$  called a *potential function*. For

$x > 0$ ,  $x \in W_x$  and  $c^\top x > 0$ , define

$$f(x, c) = n \ln(c^\top x) - \sum_{j=1}^n \ln(x_j) = \sum_{j=1}^n \ln(c^\top x / x_j). \quad (7.37)$$

This implies  $e^{f(x,c)} = (c^\top x)^n / \prod_{j=1}^n x_j \geq (c^\top x)^n$  where the inequality follows from the fact that  $\prod_{j=1}^n x_j < 1$  for all  $x > 0$  and  $x \in W_x$ . Then as the potential function  $f(x, c)$  goes to  $-\infty$  the objective function  $c^\top x$  goes to zero. Similarly, in the transformed  $y$  space, for  $y > 0$ ,  $y \in W_y$  and  $(f^k)^\top y > 0$  define

$$g(y, f^k) = n \ln((f^k)^\top y) - \sum_{j=1}^n \ln(y_j) = \sum_{j=1}^n \ln((f^k)^\top y / y_j). \quad (7.38)$$

It is left as an exercise to prove the following lemma.

**Lemma 7.8** *If  $x > 0$ ,  $x \in W_x$  and  $c^\top x > 0$ , then under the transformation  $y = X_k^{-1}x/e^\top X_k^{-1}x$ ,*

$$g(y, f^k) = f(x, c) + \sum_{j=1}^n \ln(x_j^k) \quad (7.39)$$

and for  $x = x^k$ ,

$$g(y^0, f^k) = f(x^k, c) + \sum_{j=1}^n \ln(x_j^k) \quad (7.40)$$

with  $y^0 = (1/n, \dots, 1/n)$ .

Before proving the main result we state another lemma used in its proof. The proof of the lemma does not add insight into the method under study and is omitted.

**Lemma 7.9** *If  $y \in \{y \in \mathbb{R}^n \mid (y - y_0)^\top (y - y_0) \leq (\alpha/n)^2\}$  and  $\alpha \in (0, 1)$ , then*

$$-\sum_{j=1}^n \ln(y_j) \leq -\sum_{j=1}^n \ln(1/n) + \alpha^2/2(1-\alpha)^2 \quad (7.41)$$

**Proposition 7.10** *If  $\alpha = 1/3$  and  $r = 1/n$ , then under assumptions 1 and 2  $cx^k \leq e^{-k/5n}(cx^0)$  and Karmarkar's algorithm terminates in  $5nL$  iterations.*

**Proof:** We show that the potential function in  $y$  space makes progress, specifically that it goes down by  $1/5$  at each step. At each iteration  $\bar{y} > 0$ . If  $(f^k)^\top \bar{y} = 0$  this implies  $c^\top x^{k+1} = 0$  and the result holds trivially. Assume  $(f^k)^\top \bar{y} > 0$ . Using the definition of  $g(y, f^k)$ , and combining equation (7.29) and equation (7.41) gives

$$g(\bar{y}, f^k) = n \ln((f^k)^\top \bar{y}) - \sum_{j=1}^n \ln(\bar{y}_j) \leq g(y^0, f^k) - \alpha + \frac{\alpha^2}{2(1-\alpha)^2}.$$

For  $\alpha = 1/3$  this becomes  $g(\bar{y}, f^k) \leq g(y^0, f^k) - 5/24 \leq g(y^0, f^k) - 1/5$ . Recall that at iteration  $k$  of Karmarkar's algorithm we find an optimal  $\bar{y}$  and map it back into  $x$  space by  $x^{k+1} = X_k \bar{y} / e^\top X_k \bar{y}$ . Then by Property 1 of Proposition 7.4 and Lemma 7.8,  $g(\bar{y}, f^k) = f(x^{k+1}, c) + \sum_{j=1}^n \ln(x_j^k)$ . Similarly  $x^k$  was mapped to  $y^0$  under the transformation  $X_k^{-1} x / e^\top X_k^{-1} x$ . Combining this with (7.39) and (7.40) and canceling the  $\sum_{j=1}^n \ln(x_j^k)$  term gives  $f(x^{k+1}, c) \leq f(x^k, c) - 1/5$ . We get a fixed decrease in the potential function at every iteration! Iterating back to  $x^0$  gives  $f(x^k, c) \leq f(x^0, c) - k/5$ . Using the definition of  $f(x, c)$  gives

$$n \ln(c^\top x^k) - \sum_{j=1}^n \ln(x_j^k) \leq n \ln(c^\top x^0) - \sum_{j=1}^n \ln(x_j^0) - k/5.$$

But on the simplex,  $\sum_{j=1}^n \ln(x_j^0) \geq \sum_{j=1}^n \ln(x_j^k)$ . Then  $n \ln(c^\top x^k) \leq n \ln(c^\top x^0) - k/5$ .  $\square$

**Corollary 7.11** *Given  $\alpha = 1/3$  and assumptions 1 and 2, Karmarkar's algorithm is  $O(n^4 L)$ .*

**Proof:** The proof follows directly from Proposition 7.10 and from observing that the effort to calculate  $\hat{f}^k$  in infinite precision is  $O(n^3)$  since the number of rows in the matrix  $A$  does not exceed  $n$  (it is assumed to have full row rank).  $\square$

Karmarkar [260] shows how this complexity of  $O(n^4 L)$  can be improved to  $O(n^{3.5} L)$  by updating the inverse of  $(PP^\top)$  rather than computing it from scratch at each iteration. See Section B.7 in Appendix B for an infinite precision, polynomial algorithm for Gaussian elimination.

## 7.5 PURIFICATION, STANDARD FORM AND SLIDING OBJECTIVE

### 7.5.1 Finding an Optimal Extreme Point

Assume that after  $k$  iterations Karmarkar's algorithm has terminated with  $c^\top x^k \leq 2^{-L} c^\top x^0$ . It is possible to find an optimal extreme point  $\hat{x}$  in strongly polynomial time from  $x^k$ . This process is known as *purification* or *crossover*. It is based on the following lemma which is left as an exercise.

**Lemma 7.12** *If  $x^k$  is a feasible solution to the linear program (KLP), then an extreme point solution  $\hat{x}$  to (KLP) can be found in  $O(m^2n)$  time such that  $c^\top \hat{x} \leq c^\top x^k$ .*

See Bixby and Saltzman [58] for details of how the CPLEX linear programming package finds a basic solution from an interior point solution.

**Proposition 7.13** *If  $\hat{x}$  is an extreme point found from the purification process such that  $c^\top \hat{x} \leq c^\top x^k \leq 2^{-L} c^\top x^0$  then  $c^\top \hat{x} = 0$  and  $\hat{x}$  is an optimal solution to linear program (KLP).*

**Proof:** Assume, without loss, that the objective function vector  $c$  and constraint matrix  $A$  in the Karmarkar standard form (KLP) are integral. Let  $A_B$  be the basis matrix for the extreme point  $\hat{x}$  with corresponding basis index set  $B$ . By assumption 2 the optimal value of (KLP) is zero so  $c^\top \hat{x} \geq 0$ . If  $c^\top \hat{x} > 0$  then

$$c^\top \hat{x} = \frac{c_B^\top \text{adj}(A_B)b}{\det(A_B)} = \frac{N}{|\det(A_B)|} \geq \frac{1}{|\det(A_B)|}$$

where  $\det(A_B)$  is the determinant of matrix  $A_B$ ,  $\text{adj}(A_B)$  is the adjoint matrix of  $A_B$ , and  $N$  is a positive integer. By the stopping criterion  $c^\top \hat{x} \leq 2^{-L} c^\top x^0$ , which implies

$$2^{-L} c^\top x^0 \geq (1/\det(A_B)).$$

Also, because  $x^0$  lies in an  $n-1$  dimensional simplex,  $c^\top x^0 \leq \max_{j=1,\dots,n} \{|c_j|\} \leq 2^{\langle c \rangle}$ . Then  $2^{-L} 2^{\langle c \rangle} \geq (1/\det(A_B))$  which is a contradiction of Lemma B.4.  $\square$

Proposition 7.13 guarantees that an optimal extreme point solution  $\hat{x}$  can be found in strongly polynomial time from the terminating solution  $x^k$  of Karmarkar's algorithm. However, an optimal dual solution is not necessarily at

hand. If  $A_B$  is the basis matrix associated with the optimal extreme point  $\hat{x}$ , it is not necessarily true that  $y = c_B^\top A_B^{-1}$  is dual feasible. Ye and Kojima [470] address this problem. In fact, Megiddo [325] has shown that given an optimal primal solution, if there exists a strongly polynomial algorithm to find an optimal primal-dual basis then there exists a strongly polynomial algorithm for linear programming. Shortly we will study interior point methods that generate both primal and dual optimal solutions. This is very important for sensitivity analysis.

### 7.5.2 Karmarkar Standard Form

In order to implement Karmarkar's algorithm it is necessary to put a linear program  $\min\{c^\top x \mid Ax = b, x \geq 0\}$  into the form given in formulation (KLP). First we assume that the linear program has a bounded feasible region. This is not really a big restriction because if the linear program has an optimal solution it has an optimal extreme point. It is possible to bound the size of the extreme point solution given the input  $A$  and  $b$ . Then a valid modified linear program is  $\min\{c^\top x \mid Ax = b, e^\top x \leq M, x \geq 0\}$ . It is left as an exercise to find a valid value of  $M$  as a function of  $L$  defined in equation (7.20). Add a slack variable and the linear program in  $n + 1$  variables is  $\min\{c^\top x \mid Ax = b, e^\top x + x_{n+1} = M, x, x_{n+1} \geq 0\}$ . Now we can "homogenize" this system just as we did in Chapter 3.

$$\begin{aligned} & \min c^\top x \\ \text{s.t. } & Ax - bx_0 = 0 \\ & e^\top x + x_{n+1} - Mx_0 = 0 \\ & x_0 = 1 \\ & x_0, x, x_{n+1} \geq 0 \end{aligned}$$

Now multiply the constraint  $x_0 = 1$  by  $M + 1$  and add it to the constraint  $e^\top x + x_{n+1} - Mx_0 = 0$ . This gives the following equivalent system.

$$\begin{aligned} & \min c^\top x \\ \text{s.t. } & Ax - bx_0 = 0 \\ & e^\top x + x_{n+1} + x_0 = M + 1 \\ & x_0 = 1 \\ & x_0, x, x_{n+1} \geq 0 \end{aligned}$$

Now multiply the constraint  $e^\top x + x_{n+1} + x_0 = M + 1$  by  $-1/(M + 1)$  and add it to the constraint  $x_0 = 1$  to get a new equivalent system.

$$\min c^\top x$$

$$\begin{aligned}
 \text{s.t. } Ax - bx_0 &= 0 \\
 e^T x + x_{n+1} + x_0 &= M + 1 \\
 e^T x + x_{n+1} - Mx_0 &= 0 \\
 x_0, x, x_{n+1} &\geq 0
 \end{aligned}$$

Next scale the variables by making the substitution  $z = (M + 1)x$ ,  $z_0 = (M + 1)x_0$ , and  $z_{n+1} = (M + 1)x_{n+1}$ .

$$\begin{aligned}
 \min c^T z \\
 \text{s.t. } Az - bz_0 &= 0 \\
 e^T z + z_{n+1} + z_0 &= 1 \\
 e^T z + z_{n+1} - Mz_0 &= 0 \\
 z_0, z, z_{n+1} &\geq 0
 \end{aligned}$$

This system must be massaged so that  $(1/n + 2, \dots, 1/n + 2)$  is a feasible solution. Introduce one more artificial variable  $z_{n+2}$  with a large objective function cost coefficient  $Q$  so that this variable is never positive if there is a feasible solution to the original linear program. It is left as an exercise to find  $Q$  as a function of  $L$ .

$$\min c^T z + Qz_{n+2} \quad (7.42)$$

$$\text{s.t. } Az - bz_0 - [Ae - b]z_{n+2} = 0 \quad (7.43)$$

$$e^T z + z_0 + z_{n+1} + z_{n+2} = 1 \quad (7.44)$$

$$e^T z + z_0 - Mz_{n+1} - (n + 1 - M)z_{n+2} = 0 \quad (7.45)$$

$$z_0, z, z_{n+1}, z_{n+2} \geq 0 \quad (7.46)$$

The vector  $(1/n + 3, \dots, 1/n + 3)$  is a feasible solution to this linear program. An important aspect of this modified formulation is that the density of the  $Ax = b$  system is preserved except for the columns associated with variable  $z_0$  and  $z_{n+2}$  since  $b$  can be quite dense in practice.

The linear program (7.42)-(7.46) satisfies assumption 1 for the Karmarkar standard form. It is still necessary to show that the linear program has an optimal solution value of zero. Assume that the original linear program is  $\max\{c^T x \mid Ax = b, x \geq 0\}$ . The optimality conditions for this linear program are  $Ax = b$ ,  $x \geq 0$ ,  $A^T u \leq c$ ,  $c^T x = b^T u$ . In this description of the optimality conditions we have replaced complementary slackness with the equivalent condition  $c^T x = u^T b$ . Now convert this system using the methods just described. In the resulting objective function (7.42), only the artificial variable  $z_{n+2}$  has a nonzero coefficient and the optimal solution value is zero if and only if there

is an optimal solution to the original linear program. The problem with this approach is that it greatly increases the size of the original system. Karmarkar has proposed an alternative method.

### 7.5.3 Unknown Optimal Objective Function

In most cases the optimal value of the linear program is not known. Unfortunately, assumption 2 requires that the optimal solution value is zero. A method to handle this problem is the “sliding objective function.” See Karmarkar [260]. First observe that one can, in polynomial time, check if the optimal objective function value is positive or negative.

1. The optimal objective function value  $c^\top x^*$  is nonpositive. If at some iteration  $c^\top x^{k+1} = c^\top X_k \bar{y} / e^\top X_k \bar{y} < 0$ , then the optimal objective function is negative and we terminate with that answer. However, if  $c^\top x^{k+1} > 0$  at every iteration then the potential function still decreases by  $\frac{1}{5}$  at each iteration (the proof of this is left as an exercise). Then after at most  $k = 5nL$  iterations we have a solution which satisfies  $c^\top x^k < 2^{-L}c^\top x^0$ . Using an argument identical to that used in the proof of Proposition 7.13 we show that  $c^\top x^*$  is not positive.
2. The optimal value  $c^\top x^*$  is strictly positive. Let  $A_B$  be the basis associated with an optimal extreme point that has a strictly positive objective function value. Again referring back to the proof of Proposition 7.13 we know that  $c^\top x^* > 1/\det(A_B)$ ; but, if the potential function decreases by  $\frac{1}{5}$ , after  $k = 5nL$  iterations  $c^\top x^k \leq c^\top x^0 2^{-L} \leq 1/\det(A_B)$ . Therefore there is an iteration that violates the guaranteed decrease of  $\frac{1}{5}$  of the potential function.

We have shown that the sign of the objective function is known in polynomial time. This is used as follows. From Lemma B.6, it is possible to bound the optimal objective function value by  $-2^L \leq l \leq c^\top x^* \leq u \leq 2^L$ . Define

$$\begin{aligned} l' &= l + (1/3)(u - l) \\ u' &= l + (2/3)(u - l) \end{aligned}$$

Replace the objective function  $\sum_{j=1}^n c_j x_j$ , with  $\sum_{j=1}^n c_j x_j - \sum_{j=1}^n l' x_j$ . Because of the simplex constraint  $\sum_{j=1}^n x_j = 1$ , this modified objective function will have an optimal value which is nonpositive if  $l' \geq c^\top x^*$  and a positive value if  $l' < c^\top x^*$ . Apply Karmarkar’s algorithm to the problem with this modified

objective function. Using the reasoning just given, after a polynomial number of iterations we can determine if the objective function value is nonpositive or positive. If it is nonpositive then  $l'$  is a new upper bound and we set  $u \leftarrow l'$ . If the objective function value is positive then  $l'$  is a new lower bound so  $l \leftarrow l'$ . In either case the interval of uncertainty is decreasing geometrically. After  $O(L)$  runs the range is reduced from  $2^{O(L)}$  to  $2^{-2L}$  and an optimal solution identified. The details are left as an exercise.

See also Todd and Burrell [426] for a clever method using dual information to handle the problem of an unknown objective function value.

## 7.6 AFFINE POLYHEDRAL TRANSFORMATIONS

The motivation behind Karmarkar's projective transformation was to transform the polytope so that the current iterate,  $x^k$ , is at the center of the transformed polytope. In this case it makes sense to move in the direction of the projected steepest ascent. The projective transformation used by Karmarkar is nonlinear and simpler transformations have been studied. Of particular interest are *affine transformations*. The affine transformation  $T : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is constructed so that, given the current iterate  $x^k$ ,  $T(x^k) \rightarrow e$ . Thus, the current iterate is transformed to a point which has equal distance from each of the facets that define the nonnegative orthant. The transformation used in Example 7.1 is an affine transformation. Like the projective transformations, the iterate  $x^0 = (1, 0.5, 6288, 3594.5, 2120, 2695.5)$  is transformed to the point  $(1, 1, 1, 1, 1, 1)$ . However, observe from Figures 7.1 and 7.2 that the affine transformation stretches the polytope and it is no longer the case that a simplex is transformed into a simplex. Thus, there is scaling, but no projection of the transformed point back onto the simplex. Hence, the transformation is an affine transformation, not a projective transformation. In the next two subsections we study the primal and dual affine scaling algorithms, respectively.

### 7.6.1 Primal Affine Scaling

Because the simplex structure is not preserved under affine transformations, the primal affine scaling algorithm works with the standard form linear program

$$\min c^\top x$$

$$(LP) \quad \begin{aligned} & \text{s.t. } Ax = b \\ & \quad x \geq 0. \end{aligned}$$

The dual linear program is

$$(DLP) \quad \begin{aligned} & \max \quad b^T u \\ & \text{s.t. } A^T u + w = c \\ & \quad w \geq 0 \end{aligned}$$

where  $A$  is an  $m \times n$  matrix of integers,  $c$  is an  $n$  component integer vector and  $b$  and  $m$  component integer vector. We assume that  $A$  has rank  $m$  and that there exists  $x^0 > 0$  with  $Ax^0 = b$ .

At iteration  $k$  of the algorithm, an affine transformation  $T_k : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is defined by

$$y = T_k(x) = X_k^{-1}x \quad (7.47)$$

where  $X_k$  is the diagonal matrix of positive elements

$$X_k = \begin{bmatrix} x_1^k & 0 & \cdots & 0 \\ 0 & x_2^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n^k \end{bmatrix}.$$

This transformation has several important properties.

**Lemma 7.14** If

$$X_k = \begin{bmatrix} x_1^k & 0 & \cdots & 0 \\ 0 & x_2^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n^k \end{bmatrix}$$

and  $x_i^k > 0$ ,  $i = 1, \dots, n$ , then under the transformation  $y \leftarrow X_k^{-1}x$ ,

1.  $e \leftarrow X_k^{-1}x^k$ ;
2. if  $y = X_k^{-1}x$ , then  $x = X_k y$ ;
3. if  $P \subset \mathbb{R}^n$  is a polyhedron, then  $P' \leftarrow X_k^{-1}P$  is also polyhedron of  $\mathbb{R}^n$  and if  $x$  is an interior point of  $P$  then  $y = X_k^{-1}x$  is an interior point of the polyhedron  $P'$ .

Substituting  $X_k y$  for  $x$  in the linear program ( $LP$ ) yields the transformed affine linear program  $ALP(X_k)$ .

$$(ALP(X_k)) \quad \begin{aligned} & \min c^\top X_k y \\ & \text{s.t. } AX_k y = b \\ & \quad y \geq 0 \end{aligned}$$

By property 1 of Lemma 7.14, the current iterate point  $x^k$  which is an interior point of the feasible region ( $LP$ ) is mapped into the point  $e^\top = (1, 1, \dots, 1)$  in the transformed space. Then in the transformed space we are a distance of at least 1 unit from the boundary created by the nonnegativity constraints. Just as in Karmarkar's algorithm, the philosophy is to center the current iterate within the polytope so a large step can be made in the direction of the negative of the projected gradient. Let's apply the gradient projection algorithm developed in the introduction of this chapter to the linear program ( $ALP(X_k)$ ).

The gradient of linear program ( $ALP(X_k)$ ) is  $c^\top X_k$ . This is projected onto the linear space defined by the constraints  $AX_k y = 0$  by solving the following least squares problem.

$$\min_u \|X_k c - X_k A^\top u\|^2 \quad (7.48)$$

The solution is

$$u^{k+1} = (AX_k^2 A^\top)^{-1} (AX_k^2) c. \quad (7.49)$$

By construction,  $X_k c - X_k A^\top u^{k+1}$  gives the projection of  $X_k c$  into the linear space defined by the constraints  $AX_k y = 0$ . Then the projected gradient  $\hat{f}^k$  is

$$\hat{f}^k = X_k c - X_k A^\top (AX_k^2 A^\top)^{-1} (AX_k^2) c. \quad (7.50)$$

The maximum step size in the direction of the projected gradient without violating the nonnegativity constraints,  $y_i \geq 0$ , is given by

$$\alpha_P := \alpha \left( \min_i \left\{ 1/\hat{f}_i^k \mid \hat{f}_i^k > 0 \right\} \right) \quad (7.51)$$

where  $\hat{f}_i^k$  is defined in (7.50) and  $\alpha$  is selected in the interval  $(0, 1)$ . This prevents the new point from touching the boundary. The new point in the transformed space is

$$y^{k+1} = y^k - \alpha_P \hat{f}^k. \quad (7.52)$$

This point is then transformed back into  $x$ -space by  $x^{k+1} = X_k y^{k+1}$ . The definition of  $\alpha_P$  guarantees that  $x^{k+1} > 0$  and that the new transformation

matrix  $X_{k+1}$  is nonsingular. This procedure is called *primal affine scaling*. In the space of the original linear program the search direction is

$$-X_k \hat{f}^k = -X_k(X_k c - X_k A^\top (A X_k^2 A^\top)^{-1}(A X_k^2) c). \quad (7.53)$$

The solution of the least squares problem at iteration  $k$  provides a vector of dual variables  $u^{k+1} = (A X_k^2 A^\top)^{-1}(A X_k^2) c$ . If  $w^{k+1} = c - A^\top u^{k+1} \geq 0$  then  $(x^{k+1}, u^{k+1}, w^{k+1})$  is primal-dual feasible. For this solution to be primal-dual optimal, it is necessary and sufficient that

$$c^\top x^{k+1} - b^\top u^{k+1} = (x^{k+1})^\top w^{k+1} = 0.$$

These ideas suggest the following algorithm.

### Algorithm 7.15 (Primal Affine Scaling)

**Step 1: (Initialization)** Set  $k \leftarrow 0$  and select primal solution  $x^0 > 0$  such that  $Ax^0 = b$  and dual solution  $u^0$ , ( $u^0$  is not necessarily dual feasible),  $\epsilon > 0$  and  $\alpha \in (0, 1)$ .

**Step 2: (Project - Least Squares)** Find the direction in which to move. Do this by solving the least squares problem

$$\min_u \|X_k c - X_k A^\top u\|^2 \quad (7.54)$$

and obtain the solution  $u^{k+1} \leftarrow (A X_k^2 A^\top)^{-1}(A X_k^2) c$ .

**Step 3: (Calculate the projected gradient)**

$$\hat{f}^k \leftarrow X_k c - X_k A^\top u^{k+1}$$

If  $\hat{f}^k = 0$ , stop,  $x^k$  is primal optimal. If  $\hat{f}^k \neq 0$  and  $\hat{f}^k \geq 0$ , stop, the primal is unbounded. Otherwise, go to **Step 4**.

**Step 4: (Obtain new point)**

$$x^{k+1} \leftarrow X_k y^{k+1} = X_k(e - \alpha_P \hat{f}^k)$$

where  $\alpha_P \leftarrow \alpha(\min_i \{1/\hat{f}_i^k \mid \hat{f}_i^k > 0\})$ . If  $c^\top x^k - b^\top u^k \geq \epsilon$  or  $w^k = c - A^\top u^k$  has at least one negative component, increment  $k \leftarrow k + 1$  and go to **Step 2**.

**Example 7.16** Consider the following linear program.

$$\begin{aligned}
 & \max 10x_1 + 9x_2 \\
 \text{s.t.} \quad & 7x_1 + 10x_2 + x_3 = 6300 \\
 & 3x_1 + 5x_2 + x_4 = 3600 \\
 & 3x_1 + 2x_2 + x_5 = 2124 \\
 & 2x_1 + 5x_2 + x_6 = 2700 \\
 & x_1, x_2 \geq 0
 \end{aligned}$$

The Mathematica implementation is listed below. In Figure 7.8 we show the path of solutions for 15 iterations of the algorithm through the interior of the polytope in the solution space of the original variables  $(x_1, x_2)$ . The algorithm is converging to the optimal solution of  $(540, 252)$ .

```

(* MATHEMATICA IMPLEMENTATION OF PRIMAL
AFFINE TRANSFORMATION ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
alpha = .9;
n = 6;
m = 4;
Iterlim = 15;
c={{-10.}, {-9.}, {0.},
 , {0.}, {0.}, {0.}};
A={{7., 10., 1.0, 0., 0., 0.},
 {3., 5., 0., 1., 0., 0.},
 {3., 2., 0., 0., 1., 0.},
 {2., 5., 0., 0., 0., 1.}};
b = { {6300.}, {3600.},
 {2124.}, {2700.} } ;
x={{1.0}, {1.0}, {6283.}, {3592.},
 {2119.}, {2693.}};
u={{-1.}, {-1.}, {-1.}, {-1.}};
s={{5.}, {13.}, {1.}, {1.}, {1.}, {1.}};
X = IdentityMatrix[n];
W = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = 0. ;
For[j = 1, j <= n, j++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
 For[i = 1, i <= m, i++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] ];
 ];
For[i = 1, i <= m, i++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[b[[i,1]]]]] ];

```

```

(* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-2.L);
dualitygap = Part[Transpose[x].s, 1, 1];
iterkount = 1;
While[( dualitygap > epsilon && iterkount <= Iterlim ),
For[j = 1, j <=n, j++,
  X[[j,j]] = x[[j, 1]];
  W[[j, j]] = s[[j, 1]]; ];
Part[w, iterkount, 1] = Part[x, 1, 1];
Part[w, iterkount, 2] = Part[x, 2, 1];
Print["x = ", x, " dualitygap =", dualitygap];
Print["w = ", w];
(* CALCULATE THE PRIMAL DIRECTION *)
u = Inverse[A.X.X.Transpose[A]].A.X.X.c;
Dx = X.c - X.Transpose[A].u;
(* PRIMAL SPACE RATIO TEST *)
alphaP = 1.0;
y = Inverse[X].x;
For[j = 1, j <= n, j++,
  alphaP = If[Dx[[j,1]] > 0,
    Min[alphaP, 1. / Dx[[j,1]] ],
    alphaP]; ];
alphaP = alpha*alphaP;
(* OBTAIN THE NEW POINT *)
x = x - alphaP*X.Dx;
w = c - Transpose[A].u;
(* UPDATE GAP *)
dualitygap = Part[ Transpose[x].w, 1, 1 ];
(* CHECK DUAL FEASIBILITY *)
flag = 1;
For[j = 1, j <= n, j++,
  flag = If[w[[j, 1]] >= 0., flag, 0 ]; ];
iterkount = iterkount + 1;
dualitygap = If[flag >= 1, dualitygap, Infinity]; ]
(* END THE WHILE STATEMENT *)

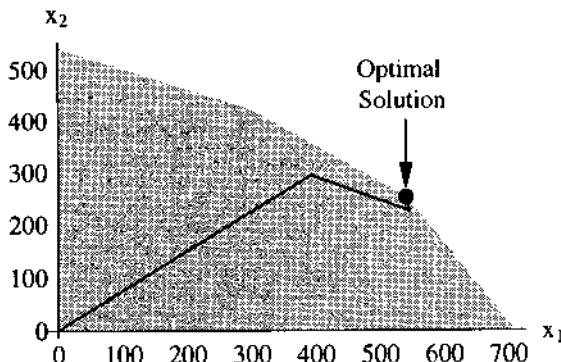
```

Unlike Karmarkar's algorithm, primal affine scaling is strictly monotonic decreasing. Since  $x^{k+1} = X_k(e - \alpha_P \hat{f}^k) = x^k - \alpha_P X_k \hat{f}^k$

$$c^\top x^{k+1} = c^\top x^k - \alpha_P c^\top X_k \hat{f}^k.$$

By definition of  $\hat{f}^k$ ,  $c^\top X_k$  is the sum of  $\hat{f}^k$  and a vector orthogonal to  $\hat{f}^k$ , which implies

$$c^\top x^{k+1} = c^\top x^k - \alpha_P (\hat{f}^k)^\top \hat{f}^k = c^\top x^k - \alpha_P \|\hat{f}^k\|^2$$

**Figure 7.8** Solution Path From Primal Affine Scaling Algorithm

and either  $\hat{f}^k = 0$  in which case  $x^k$  is optimal, or  $\hat{f}^k \neq 0$  and  $c^T x^{k+1} < c^T x^k$ .

The primal affine scaling algorithm is due to Dikin [123] and was later rediscovered by Barnes [36] and Vanderbei, Meketon, and Freedman [441].

Since the sequence  $\{c^T x^k\}$  is monotonic decreasing, if the linear program has an optimal solution then the sequence  $\{c^T x^k\}$  is bounded below and therefore converges. However, this does not imply that the sequence  $\{x^k\}$  converges to an optimal solution of  $(LP)$ . Convergence of  $\{x^k\}$  to an optimal solution can be shown with appropriate nondegeneracy assumptions and conditions on the primal feasible region being bounded. A number of authors have addressed the convergence issue. See, for example, Dikin [124], Hall and Vanderbei [221], Monteiro, Tsuchiya, and Wang [344], Saigal [391] and Vanderbei and Lagarias [444]. Tsuchiya [434] provides convergence results without nondegeneracy assumptions. This result is strengthened in Tsuchiya and Muramatsu [435] where convergence is shown for long step sizes, i.e.,  $\alpha_P$  up to  $2/3$ .

Unfortunately, to date, no one has shown that the primal affine scaling can be used to find an optimal solution to  $(LP)$  in polynomial time. Work by Megiddo and Shub [326] suggest that this may not be possible.

## 7.6.2 Dual Affine Scaling

The dual affine scaling algorithm is due to Adler et al. [3]. The basic approach is to apply the affine scaling idea of the previous subsection to the dual

linear program (*DLP*). Instead of centering the primal variables  $x$  at each iteration we center the dual variables  $w$  at each iteration. Again, define an affine transformation  $T_k : \mathbb{R}^n \rightarrow \mathbb{R}^n$  by

$$v = T_k(w) = W_k^{-1}w \quad (7.55)$$

where  $W_k$  is the diagonal matrix of positive elements

$$W_k = \begin{bmatrix} w_1^k & 0 & \cdots & 0 \\ 0 & w_2^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n^k \end{bmatrix}.$$

Substituting  $W_k v$  for  $w$  in the linear program (*DLP*) yields the transformed linear program *ADLP*( $W_k$ ).

$$\begin{aligned} \text{i.t. } & \max && b^\top u \\ \text{ADLP}(W_k) & \text{s.t. } && A^\top u + W_k v = c \\ & && v \geq 0 \end{aligned}$$

The projection of the feasible region defined of *ADLP*( $W_k$ ) onto the space of slack variables is  $V = \{v \geq 0 \mid \text{there exists } u \text{ such that } A^\top u + W_k v = c\}$ . When  $A^\top$  is full rank, Gonzaga [196] has observed that for any  $\bar{v} \in V$  there is a unique  $\bar{u}$  such that  $A^\top \bar{u} + W_k \bar{v} = c$  and  $\bar{u}$  is given by

$$\bar{u} = (AW_k^{-2}A^\top)^{-1}AW_k^{-1}(W_k^{-1}c - \bar{v}). \quad (7.56)$$

Now *ADLP*( $W_k$ ) can be expressed as a linear program in the slack variables only.

$$\max -b^\top (AW_k^{-2}A^\top)^{-1}AW_k^{-1}v \quad (7.57)$$

$$\begin{aligned} \text{s.t. } & (I - W_k^{-1}A^\top(AW_k^{-2}A^\top)^{-1}AW_k^{-1})v \\ & = (I - W_k^{-1}A^\top(AW_k^{-2}A^\top)^{-1}AW_k^{-2})c \end{aligned} \quad (7.58)$$

$$v \geq 0 \quad (7.59)$$

The gradient of the objective function is  $-W_k^{-1}A^\top(AW_k^{-2}A^\top)^{-1}b$ . Fortunately, the gradient lies in the linear space of the equality constraints (7.58). This implies that the direction of steepest ascent is also a feasible direction. Then the direction in  $v$ -space is

$$\Delta v = -W_k^{-1}A^\top(AW_k^{-2}A^\top)^{-1}b. \quad (7.60)$$

This implies the direction in  $u$ -space because any feasible direction  $(\Delta u, \Delta v)$  must satisfy  $A^\top \Delta u + W_k \Delta v = 0$ . Again using the fact that  $A^\top$  has full rank, this implies

$$\Delta u = -(AW_k^{-2}A^\top)^{-1}AW_k^{-1}\Delta v.$$

This, together with (7.60) gives

$$\Delta u = -(AW_k^{-2}A^\top)^{-1}AW_k^{-1}(-W_k^{-1}A^\top(AW_k^{-2}A^\top)^{-1}b) \quad (7.61)$$

$$= (AW_k^{-2}A^\top)^{-1}b \quad (7.62)$$

The maximum step size from  $e$  in the direction  $\Delta v$  without violating the non-negativity constraints,  $v_i \geq 0$ , is given by

$$\alpha_D := \alpha(\min_i \{1/(-\Delta v_i) \mid \Delta v_i < 0\}) \quad (7.63)$$

where  $\alpha$  is selected in the interval  $(0, 1)$ . This prevents the new point from touching the boundary. The new solution in the transformed space is

$$\begin{aligned} u^{k+1} &= u^k + \alpha_D \Delta u \\ v^{k+1} &= v^k + \alpha_D \Delta v. \end{aligned}$$

This point is then transformed back into the original  $(u, w)$  space by  $w^{k+1} = W_k v^{k+1}$ . The definition of  $\alpha_D$  guarantees that  $w^{k+1} > 0$  and the new transformation matrix  $W_{k+1}$  is nonsingular. This procedure is called *dual affine scaling*.

As in the case of primal affine scaling, finding the ascent directions provides dual information (which are the original primal variables since this is the dual problem). At iteration  $k$ , the direction in  $v$ -space is given by (7.60) and it is easy to see that if  $x^{k+1} = -W_k^{-1}\Delta v$ , then  $Ax^{k+1} = b$ . It is not necessarily the case that  $x^{k+1} \geq 0$ . However, if  $x^{k+1} \geq 0$  then the primal-dual solution  $(x^{k+1}, u^{k+1}, w^{k+1})$  is primal-dual optimal if and only if

$$c^\top x^{k+1} - b^\top u^{k+1} = (x^{k+1})^\top w^{k+1} = 0.$$

These ideas suggest the following algorithm.

#### Algorithm 7.17 (Dual Affine Scaling)

**Step 1: (Initialization)** Set  $k \leftarrow 0$  and select  $w^0 > 0$  and  $u^0$  such that  $A^\top u^0 + w_0 = b$  and  $x^0$  ( $x^0$  is not necessarily primal feasible),  $\epsilon > 0$  and  $\alpha \in (0, 1)$ .

**Step 2: (Calculate Direction)** Find the direction in which to move.

$$\begin{aligned}\Delta v &\leftarrow -W_k^{-1}A^T(AW_k^{-2}A^T)^{-1}b \\ \Delta u &\leftarrow (AW_k^{-2}A^T)^{-1}b\end{aligned}$$

If  $\Delta v = 0$ , stop,  $(u^k, w^k)$  is dual optimal. If  $\Delta v \neq 0$  and  $\Delta v \geq 0$ , stop, the dual is unbounded. Otherwise, go to Step 3.

**Step 3: (Obtain new point)**

$$\begin{aligned}u^{k+1} &\leftarrow u^k + \alpha_D \Delta u \\ v^{k+1} &\leftarrow v^k + \alpha_D \Delta v \\ w^{k+1} &\leftarrow W_k v^{k+1}\end{aligned}$$

where  $\alpha_D \leftarrow \alpha(\min_i \{1/(-\Delta v_i) | \Delta v_i < 0\})$

If  $c^T x^k - b^T u^k \geq \epsilon$  or  $x^k$  not primal feasible increment  $k \leftarrow k + 1$  and go to Step 2.

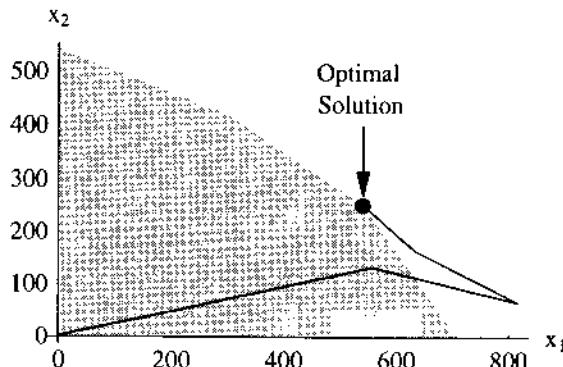
**Example 7.18** Refer back to the linear program in Example 7.16 and apply the dual affine scaling algorithm. The Mathematica implementation is listed below. In Figure 7.9 we show the path of solutions of the algorithm through the interior of the polytope in the solution space of the original variables  $(x_1, x_2)$ . Notice that with dual affine scaling primal feasibility is not maintained and we jump out of the primal polytope and then back in.

```
(* MATHEMATICA IMPLEMENTATION OF DUAL
AFFINE TRANSFORMATION ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
alpha = .9;
n = 6;
m = 4;
Iterlim = 10;
c = {{-10.}, {-9.}, {0.},
      {0.}, {0.}, {0.}};
A = {{7., 10., 1.0, 0., 0., 0.},
     {3., 5., 0., 1., 0., 0.},
     {3., 2., 0., 0., 1., 0.},
     {2., 5., 0., 0., 0., 1.}};
b = {{6300.}, {3600.},
     {2124.}, {2700.}};
x = {{1.0}, {1.0}, {6283.}, {3592.},
```

```

{2119.}, {2693.});
u={{-1.}, {-1.}, {-1.}, {-1.}};
w={{5.}, {13.}, {1.}, {1.}, {1.}, {1.}};
W = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = 0.;
For[j = 1, j <= n, j++,
  L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
  For[i = 1, i <= m, i++,
    L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] - 1];
  ];
For[i = 1, i <= m, i++,
  L = L + Ceiling[1. + Log[2, 1. + Abs[b[[i, 1]]]]] - 1];
(* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-2.L);
dualitygap = Part[Transpose[x].s, 1, 1];
iterkount = 1;
While[( dualitygap > epsilon && iterkount <= Iterlim ),
  For[j = 1, j <= n, j++,
    W[[j, j]] = w[[j, 1]] - 1;
  v = Inverse[W].w;
  Part[w, iterkount, 1] = Part[x, 1, 1];
  Part[w, iterkount, 2] = Part[x, 2, 1];
  Print["x = ", x, " dualitygap = ", dualitygap];
  Print["w = ", w];
  (* CALCULATE THE DUAL DIRECTION *)
  Dv = -Inverse[W].Transpose[A].Inverse[A].Inverse[W].Inverse[W].Transpose[A].b;
  Du = Inverse[A].Inverse[W].Inverse[W].Transpose[A].b;
  (* DUAL SPACE RATIO TEST *)
  alphaD = 1.0;
  For[j = 1, j <= n, j++,
    alphaD = If[Dv[[j, 1]] < 0,
      Min[alphaD, 1. / -Dv[[j, 1]]],
      alphaD];
  ];
  alphaD = alpha*alphaD;
  (* OBTAIN THE NEW POINT *)
  u = u + alphaD*Du;
  v = v + alphaD*Dv;
  w = W.v;
  x = -Inverse[W].Dv;
  (* UPDATE GAP *)
  dualitygap = Part[Transpose[x].s, 1, 1];
  (* CHECK PRIMAL FEASIBILITY *)
  flag = 1;
  For[j = 1, j <= n, j++,
    flag = If[x[[j, 1]] >= 0., flag, 0];
  ];
]

```

**Figure 7.9** Solution Path From Dual Affine Scaling Algorithm

```

iterkount = iterkount + 1;
dualitygap = If[flag >= 1, dualitygap, Infinity];
(*Print[dualitygap, "    flag = ", flag];*) ]
(* END THE WHILE STATEMENT *)

```

As with primal affine scaling, polynomial termination of the dual affine scaling algorithm has not been shown. The development in this section is from Adler et al. [3]. Computational results with the dual affine scaling algorithm are in Adler et al. [3] and Marsten et al. [311].

## 7.7 GEOMETRY OF THE LEAST SQUARES PROBLEM

Many of the interior point algorithms require the solution of a least squares problem. In this section we investigate the geometry of these problems. Consider the primal problem ( $ALP(X_k)$ ) derived from ( $LP$ ) through an affine transformation. The projection of the objective function gradient  $X_k c$  onto the linear space  $AX_k = 0$  is found by solving the following least squares problem.

$$\min_u \|X_k c - X_k A^T u\|^2.$$

In the primal space, if  $H = \{y \mid AX_k y = 0\}$ , then for any  $u$ , the vector  $X_k A^T u$  is in  $H^\perp$ . The solution to the least squares problem gives a  $u^{k+1}$  such that

the Euclidian distance between the vector  $X_k A^\top u$  in  $H^\perp$  and the gradient vector  $X_k c$  is minimized. In order to minimize this distance the vector  $X_k c - X_k A^\top u^{k+1}$  should be orthogonal to the linear space  $H^\perp$ .

It is now insightful to view the least squares problem in terms of the column space geometry. The column space geometric interpretation is due to Dantzig and is found in [108]. More recently, Stone and Tovey [414] use the Dantzig geometry to better understand interior point methods. What follows is primarily from Stone and Tovey. In order to make the exposition cleaner we assume, without loss, that the original linear program ( $LP$ ) has the Karmarkar standard form structure ( $KLP$ ). No assumption is made about the optimal objective function value. After the affine transformation of ( $KLP$ ) the linear program ( $AKLP(X_k)$ ) is

$$(AKLP(X_k)) \quad \begin{aligned} & \min c^\top X_k y \\ & \text{s.t. } AX_k y = b \\ & \quad e^\top X_k y = 1 \\ & \quad y \geq 0 \end{aligned}$$

The interesting space to view this problem in, is the column space geometry. The columns (not including  $e^\top x = 1$ ) of the original linear program ( $KLP$ ) are

$$\left[ \begin{array}{c} A \\ c^\top \end{array} \right]$$

Because of the convexity constraint,  $\sum_{i=1}^n x_i = 1$  and the nonnegativity constraints  $x_i \geq 0$ ,  $i = 1, \dots, n$ , the linear program ( $KLP$ ) corresponds to finding a nonnegative weighting of the  $x_i$  which sum to one such that

$$\left[ \begin{array}{c} A_{\bullet 1} \\ c_1 \end{array} \right] x_1 + \left[ \begin{array}{c} A_{\bullet 2} \\ c_2 \end{array} \right] x_2 + \dots + \left[ \begin{array}{c} A_{\bullet n} \\ c_n \end{array} \right] x_n = \left[ \begin{array}{c} 0 \\ z \end{array} \right]$$

and  $z_0$  is minimized. In  $m+1$  space, plot the columns  $A_{\bullet i}$  using the coordinates  $u_1, \dots, u_m$  and the  $c_i$  in the  $z_0$  axis. This is illustrated in Figure 7.10 for the linear program

$$\begin{aligned} & \min x_2 + 2x_3 \\ & \text{s.t. } x_1 - 3x_2 + 2x_3 = 0 \\ & \quad x_1 + x_2 + x_3 = 1 \\ & \quad x_1, x_2, x_3 \geq 0. \end{aligned}$$

The optimal solution to the linear program (*KLP*) is given by finding a convex combination of the  $A_{\bullet i}$  which intersect the  $z_0$  axis at the lowest point. Now consider this geometry from the viewpoint of a statistician (as Dantzig is by training). The formula for a straight line in this space is  $z_0 = v + u^\top w$  where  $v$  is the  $z_0$  intercept and  $w$  is an  $m$  vector of variables. If a column  $[A_{\bullet i}, c_i]^\top$  is on the line  $z_0 = v + u^\top w$ , then  $c_i = v + u^\top A_{\bullet i}$ , otherwise there is a residual of  $c_i - v - u^\top A_{\bullet i}$ . The least squares problem is to minimize the *weighted* sum of the residuals squared, i.e.,

$$\min_u \|X_k c - X_k A^\top u\|^2 = \min \sum_{i=1}^n (x_i^k(c_i - u^\top A_{\bullet i} - v))^2.$$

This means that the weighted residuals correspond to the components of the projected gradient vector. If a weighted residual is negative, i.e.  $x_i^k(c_i - u^\top A_{\bullet i} - v) < 0$  there is motivation to “push” the regression line down in this direction since we want the  $z_0$  intercept to be as low as possible and still be a convex combination of the  $A_{\bullet i}$ . This is exactly what happens in the affine scaling algorithm because the new direction in  $y$  space is the negative of the projection of  $X_k c$  onto the null space  $\{y \mid AX_k y = 0, e^\top X_k y = 0\}$ . After moving in this direction, a point  $y^{k+1}$  is obtained and the  $i$ th component of  $y^{k+1}$  is greater than the  $i$ th component of  $y^k$ . Since  $y^{k+1}$  is transformed back into  $x$ -space by  $x^{k+1} = X_k y^{k+1}$  increasing the  $i$ th component of  $y$  has the effect of increasing the  $i$ th component of  $x$ . This in turns yields a new least squares problem based on the new weights  $X_{k+1}$  where more weight is given to the components that had negative residuals at the previous iteration. Similarly, components with positive residuals at the previous iteration now have less weight. The net effect is to push the regression line down. This is illustrated in Example 7.19.

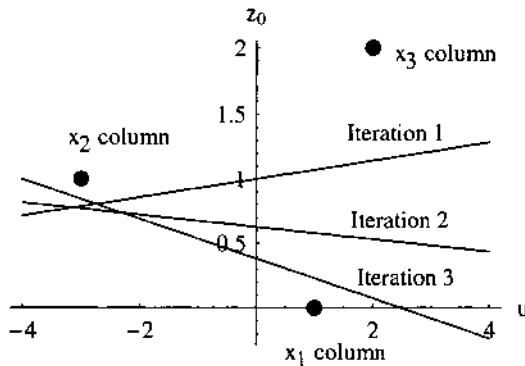
**Example 7.19** Consider the linear program

$$\begin{aligned} & \min x_2 + 2x_3 \\ \text{s.t. } & x_1 - 3x_2 + 2x_3 = 0 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

In Figure 7.10 we show the regression line for the first three iterations of the affine scaling algorithm.

**Iteration 1:**

Figure 7.10 Column Space Geometry of Least Squares Problem



```

Weights = {{0.333333}, {0.333333}, {0.333333}}
Residuals = {{-1.07143}, {0.214286}, {0.857143}}
Slope = 0.0714286
z_{0} Intercept = 1

Iteration 2:
Weights = {{0.440476}, {0.311905}, {0.247619}}
Residuals = {{-0.579725}, {0.231235}, {1.46754}}
Slope = -0.0472602
z_{0} Intercept = 0.626985

Iteration 3:
Weights = {{0.541706}, {0.291659}, {0.166635}}
Residuals = {{-0.227804}, {0.15717}, {1.92595}}
Slope = -0.153757
z_{0} Intercept = 0.38156

```

*Notice how the  $z_0$  intercept is decreasing at each iteration and negative (positive) residuals result in greater (smaller) weights at the next iteration.*

## 7.8 CONCLUSION

We are not aware of any commercial implementations of Karmarkar's projective algorithm which are competitive with the simplex algorithm. The most successful implementations of interior point algorithms are the affine scaling variants and the barrier methods described in the next chapter. However, many of the ideas behind the projective transformation algorithm have been key in the development of the affine scaling and barrier methods. In particular, consider Karmarkar's use of a potential function with the term  $\sum_{j=1}^n \ln(x_j)$ . This term appears in the barrier methods and numerous authors have proved polynomial termination for affine scaling algorithms using Karmarkar-like potential functions. See, for example, Freund [160], Gonzaga [198] and Ye [467]. It was the projective algorithm of Karmarkar that led to the development of interior point codes which are competitive with, or exceed, the performance of simplex for large problems.

## 7.9 EXERCISES

- 7.1 Perform two iterations of Karmarkar's algorithm on the linear program

$$\begin{aligned} & \min x_1 \\ \text{s.t. } & x_1 + x_2 - 2x_3 = 0 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

- 7.2 The standard form of the linear program for Karmarkar's algorithm requires  $Ax = 0$ . Why is it necessary to have  $Ax = 0$  instead of  $Ax = b$ ?
- 7.3 Prove Lemma 7.12. Prove the Lemma by showing through simplex pivoting how to construct a basic feasible solution  $\hat{x}$  from  $\bar{x}$  such that  $c^\top \hat{x} \leq c^\top \bar{x}$ . The number of pivots required should be a polynomial function of the number of variables in the problem.
- 7.4 Prove that the maximum radius of an  $(n - 1)$  sphere with center at  $(1/n, \dots, 1/n)$  that can be inscribed in the simplex

$$S = \{(x_1, \dots, x_n) \mid \sum_{i=1}^n x_i = 1, x_i \geq 0, i = 1, \dots, n\}$$

is  $r = 1/\sqrt{n(n-1)}$  and that the radius of the minimum circumscribing sphere is  $R = \sqrt{\frac{n-1}{n}}$ .

7.5 If  $H = \{x \in \mathbb{R}^n \mid Ax = 0\}$ , prove that the projection of  $c \in \mathbb{R}^n$  onto  $H$  is given by  $c - A^\top \hat{u}$  where  $\hat{u}$  is the optimal solution to  $\min_u \|c - A^\top u\|^2$  and  $\hat{u} = (AA^\top)^{-1}Ac$ .

7.6 Assume that the  $m \times n$  matrix  $A$  has rank  $m$ . Prove that  $AA^\top$  is nonsingular.

7.7 Consider the optimization problem

$$\begin{aligned} & \min \sum_{j=1}^n c_j x_j \\ \text{s.t. } & \sum_{j=1}^n (x_j - \hat{x}_j)^2 \leq r^2 \end{aligned}$$

Prove that if  $x^0 = \hat{x}$  then  $x^1 = x^0 - rc/\|c\|$  is the optimal solution. Explain why the result does not hold if the starting point  $x^0$  is not the center of the sphere, i.e.  $x^0 \neq \hat{x}$ .

7.8 Prove Proposition 7.3.

7.9 Prove that the optimal solution to the problem

$$\max \left\{ \sum_{j=1}^n \ln(x_j) \mid \sum_{j=1}^n x_j = 1, x_j \geq 0, j = 1, \dots, n \right\}$$

is the vector  $(1/n, 1/n, \dots, 1/n)$ .

7.10 In the Karmarkar canonical form what problem would be caused by replacing the  $\sum_{j=1}^n x_j = 1$  constraint with  $\sum_{j=1}^n x_j \leq 1$ ?

7.11 Show that the  $Q$  and  $M$  in linear program (7.42)-(7.46) can be selected so that their binary coding is a linear function of the binary encoding of  $A, b$  and  $c$  in the original linear program.

7.12 Take the linear program

$$\begin{aligned} & \max 10x_1 + 9x_2 \\ \text{s.t. } & (7/10)x_1 + x_2 \leq 630 \\ & (1/2)x_1 + (5/6)x_2 \leq 600 \\ & x_1 + (2/3)x_2 \leq 708 \\ & (1/10)x_1 + (1/4)x_2 \leq 135 \\ & x_1, x_2 \geq 0 \end{aligned}$$

and convert it into an equivalent linear program that satisfies assumptions 1 and 2.

- 7.13 How is an unbounded linear program recognized using Karmarkar's algorithm? How is an infeasible linear program recognized using Karmarkar's algorithm?
- 7.14 Prove that if Karmarkar's algorithm is applied to a linear program ( $KLP$ ) whose optimal objective function value is nonpositive, then at some iteration  $cx^{k+1} < 0$ , or the usual decrease of  $\frac{1}{5}$  is found in the potential function.
- 7.15 Assume that the linear program  $\min\{cx \mid Ax = b, x \geq 0\}$  has an optimal solution. Then it is possible to make the feasible region bounded by adding the constraint  $e^T x \leq M$  for sufficiently large  $M$ . Derive a valid value for  $M$  in terms of the binary encoding of the coefficients in the linear program.
- 7.16 Prove Lemma 7.8.
- 7.17 Prove property 3 of Lemma 7.14.
- 7.18 When the optimal solution value of the linear program is not known the sliding objective function method of Karmarkar requires lower and upper bounds,  $l, u$  on the objective function value. Show that the structure of problem ( $KLP$ ) can be used to find values for  $l$  and  $u$  that are better than  $-2^L$  and  $2^L$ .
- 7.19 Give an algorithm for constructing a feasible solution to the dual of ( $KLP$ ).
- 7.20 With the sliding objective function method, the interval of uncertainty is reduced until  $u' - l' \leq 2^{-2L}$ . If  $u' - l' \leq 2^{-2L}$ , show how to find the optimal solution to ( $KLP$ ) in  $O(nL)$  iterations of Karmarkar's algorithm.
- 7.21 Show that the gradient of the objective function,  $-W_k^{-1}A^\top(AW_k^{-2}A^\top)^{-1}b$ , for the linear program (7.57)-(7.59) lies in the null space of the constraints defined by (7.58).

---

# INTERIOR POINT ALGORITHMS: BARRIER METHODS

## 8.1 INTRODUCTION

In Chapter 7 the solution philosophy was based on an affine or projective transformation so that at the start of each iteration we were at the “center” of the polytope instead of on the boundary. This allowed a large step in the direction of a projected gradient. Another “centering” philosophy used to keep a solution from being “too close” to the boundary has its roots in the barrier methods of nonlinear programming. See Frisch [161] and Fiacco and McCormick [147] for a thorough discussion. No special structure is assumed for the primal linear program and it has the standard form

$$(LP) \quad \begin{aligned} & \min c^T x \\ \text{s.t. } & Ax = b \\ & x \geq 0 \end{aligned}$$

with the dual

$$(DLP) \quad \begin{aligned} & \max b^T u \\ \text{s.t. } & A^T u + w = c \\ & w \geq 0 \end{aligned}$$

where  $A$  is an  $m \times n$  matrix of integers,  $c$  is an  $n$  component integer vector and  $b$  an  $m$  component integer vector. Denote the binary encoding of the primal by  $L$ . See Appendix B.

For the primal problem  $(LP)$ , at the boundary of the polyhedron, at least one  $x_i = 0$ . One method to prevent a descent algorithm from hitting the boundary

is to set up a barrier in the form of a *barrier function*. One such function is  $\ln(x_i)$ . As  $x_i \rightarrow 0$ ,  $\ln(x_i) \rightarrow -\infty$ . If  $\mu > 0$  then the barrier function modification of  $(LP)$  is

$$(LP_\mu) \quad \begin{aligned} \min \quad & c^\top x - \mu \sum_{j=1}^n \ln(x_j) \\ \text{s.t.} \quad & Ax = b \\ & x > 0 \end{aligned}$$

In  $(LP_\mu)$ ,  $\mu$  is the *barrier parameter*.

**Example 8.1** Recall the two variable linear program from Example 7.1 in Chapter 7.

$$\begin{aligned} \min \quad & -10x_1 - 9x_2 \\ \text{s.t.} \quad & 7x_1 + 10x_2 \leq 6300 \\ & 3x_1 + 5x_2 \leq 3600 \\ & 3x_1 + 2x_2 \leq 2124 \\ & 2x_1 + 5x_2 \leq 2700 \\ & x_1, x_2 \geq 0 \end{aligned}$$

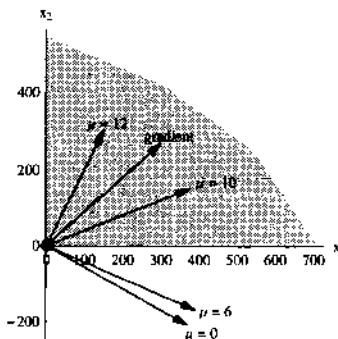
Converting to standard form gives

$$\begin{aligned} \min \quad & -10x_1 - 9x_2 \\ \text{s.t.} \quad & 7x_1 + 10x_2 + x_3 = 6300 \\ & 3x_1 + 5x_2 + x_4 = 3600 \\ & 3x_1 + 2x_2 + x_5 = 2124 \\ & 2x_1 + 5x_2 + x_6 = 2700 \\ & x_1, x_2 \geq 0 \end{aligned}$$

An initial feasible point is  $x^0 = (1, 0.5, 6288, 3594.5, 2120, 2695.5)$  with objective function value -14.5. The gradient of the objective function is  $c = [10 \ 9 \ 0 \ 0 \ 0 \ 0]^\top$ . The projected gradient,  $\hat{c}$ , on the null space  $\{x \mid Ax = 0\}$  is

$$\hat{c} = (I - A^\top (A A^\top)^{-1} A) c = \begin{bmatrix} -0.668405 \\ 0.377477 \\ 0.904067 \\ 0.117831 \\ 1.25026 \\ -0.550574 \end{bmatrix}.$$

Because the  $x_2$  component of the projected gradient is 0.377477 and the current value of  $x_2$  is 0.5, it is not possible to move very far in the direction of

**Figure 8.1** Effect of Barrier Parameter on Projected Gradients

the steepest descent (the negative projected gradient) before hitting the boundary facet  $x_2 \geq 0$ . However, if a barrier term  $-\mu \sum_{j=1}^6 \ln(x_j)$  is added to the objective function the projected gradient is “pushed” away from the  $x_2 \geq 0$  boundary. With the barrier term added the gradient of the objective function is  $(c - \mu X^{-1} e)$  (Recall that  $X^{-1}$  is the  $n \times n$  diagonal matrix with diagonal elements  $1/x_i$ .) and the projected gradient is  $(I - A^T(AA^T)^{-1}A)(c - \mu X^{-1}e)$ . Below are the projected gradients for  $\mu = 0, 6, 10$  and  $12$ .

$$\begin{aligned}(I - A^T(AA^T)^{-1}A)(c - 0X^{-1}e) &= [-0.668405 \ 0.377477 \ 0.904067 \ 0.117831 \ 1.25026 \ -0.550574]^T \\(I - A^T(AA^T)^{-1}A)(c - 6X^{-1}e) &= [-0.37402 \ 0.108456 \ 1.53358 \ 0.579779 \ 0.905147 \ 0.20576]^T \\(I - A^T(AA^T)^{-1}A)(c - 10X^{-1}e) &= [-0.177763 \ -0.0708911 \ 1.95325 \ 0.887745 \ 0.675072 \ 0.709982]^T \\(I - A^T(AA^T)^{-1}A)(c - 12X^{-1}e) &= [-0.0796349 \ -0.160565 \ 2.16309 \ 1.04173 \ 0.560034 \ 0.962093]^T\end{aligned}$$

These projected gradients after normalization are illustrated in Figure 8.1. As  $\mu$  increases the projected gradient is rotating in a counterclockwise direction. The negative of the unprojected objective function gradient is labeled “gradient.”

Problem  $(LP_\mu)$  is a nonlinear minimization problem with a strictly convex objective function. As long as  $\mu > 0$ , the optimal solution for  $(LP_\mu)$  will never have an  $x_i = 0$ , hence we write the nonnegativity constraints as  $x > 0$  rather than  $x \geq 0$ . By strict convexity, if there is an optimal solution to  $(LP_\mu)$ , then it is a unique global optimum. We are interested in characterizing when an optimal solution to  $(LP_\mu)$  exists. Clearly a primal feasible solution  $x > 0$  cannot be an optimal solution of  $LP_\mu$  if there is a direction  $\Delta x$  such that  $A\Delta x = 0$  and the inner product of the gradient of the objective function of

$(LP_\mu)$  with  $\Delta x$  is negative. If there were such a direction  $\Delta x$ , then a small move in this direction would be primal feasible and would reduce the objective function value. That is, if  $x$  is an optimal solution to  $(LP_\mu)$  there cannot be a solution to the system

$$(c - \mu X^{-1}e)^\top \Delta x < 0 \quad (8.1)$$

$$A\Delta x = 0 \quad (8.2)$$

where  $(c - \mu X^{-1}e)^\top$  is the gradient of the objective function  $c^\top x - \mu \sum_{j=1}^n \ln(x_j)$ . Recall Corollary 2.21 to Farkas' Lemma in Chapter 2. It follows from Corollary 2.21, that there is no solution to the system (8.1)-(8.2) if and only if there is a solution to the system

$$(c - \mu X^{-1}e)u_0 - A^\top u = 0, \quad u_0 > 0 \quad (8.3)$$

where we can assume without loss that  $u_0 = 1$ . Therefore, if a solution  $x$  is an optimal solution to  $(LP_\mu)$  there is a corresponding dual solution  $u$  such that the primal-dual pair  $(x, u)$  satisfies the system

$$c - \mu X^{-1}e - A^\top u = 0 \quad (8.4)$$

$$Ax = b \quad (8.5)$$

$$x > 0. \quad (8.6)$$

Equations (8.4)-(8.6) are known as the *Karush-Kuhn-Tucker optimality conditions* and characterize the global optimum if it exists. See Karush [261] and Kuhn and Tucker [274]. These are necessary conditions which are also sufficient conditions in the case of a convex optimization problem such as  $LP_\mu$ . The reader interested in a more thorough development of the Karush-Kuhn-Tucker conditions should consult a nonlinear programming text such as Bazaar, Sherali, and Shetty [44] or Luenberger [296]. Observe that condition (8.4) is really nothing more than the dual feasibility constraint  $A^\top u + w = c$  with  $w = \mu X^{-1}e$ . Make this explicit substitution and write the Karush-Kuhn-Tucker conditions as

$$A^\top u + w = c \quad (8.7)$$

$$Ax = b \quad (8.8)$$

$$x > 0 \quad (8.9)$$

$$w = \mu X^{-1}e. \quad (8.10)$$

The following logic, lemmas and proposition are taken from Monteiro and Adler [341] and Megiddo [324], and characterize the optimal solution to  $LP_\mu$ .

**Lemma 8.2** *If problem  $(LP)$  has a feasible solution then the set of optimal solutions to  $(LP)$  is bounded and not empty if and only if there is a solution  $(\bar{u}, \bar{w})$  with  $\bar{w} > 0$  to the dual problem  $(DLP)$ .*

**Lemma 8.3** *If there exists a feasible solution  $\bar{x} > 0$  to the primal problem  $(LP)$ , then for any  $\mu > 0$ , problem  $(LP_\mu)$  has an optimal solution if and only if the set of optimal solutions to  $(LP)$  is bounded and not empty.*

In order to satisfy the hypotheses of Lemma 8.2 and Lemma 8.3 the following feasibility assumptions, along with an assumption about the rank of  $A$ , are used throughout the chapter.

#### Barrier Assumptions

1. The set  $\{x \in \mathbb{R}^n \mid Ax = b, x > 0\}$  is not empty.
2. The set  $\{(u, w) \in \mathbb{R}^m \times \mathbb{R}^n \mid A^\top u + w = c, w > 0\}$  is not empty.
3. The constraint matrix  $A$  has rank  $m$ .

**Proposition 8.4** *Given Barrier Assumptions 1-3, and  $\mu > 0$ , there is a unique solution  $(x(\mu), u(\mu), w(\mu))$  to the Karush-Kuhn-Tucker conditions (8.7)-(8.10) and  $x(\mu)$  is the optimal solution to problem  $(LP_\mu)$ .*

**Proof:** Consider any  $\mu > 0$ . By Assumptions 1 and 2 there is a primal-dual feasible solution  $(\bar{x}, \bar{u}, \bar{w})$  such that  $\bar{x} > 0$  and  $\bar{w} > 0$ . Then by Lemma 8.2 there is an optimal solution to the linear program  $(LP)$  and the set of optimal solutions is bounded. Then by Lemma 8.3 there is an optimal solution  $x(\mu)$  to  $(LP_\mu)$ . By strict convexity of the objective function  $x(\mu)$  is the unique optimal solution. This implies that the solution  $w$  which satisfies (8.10) is also unique. Call this solution  $w(\mu)$ . By Barrier Assumption 3, the rank of  $A^\top$  is  $m$  which implies that there is a unique solution  $u(\mu)$  to  $A^\top u + w(\mu) = c$ .  $\square$

The proof of the following proposition (for more general problems) is found in Fiacco and McCormick [147] or Megiddo [324].

**Proposition 8.5** *Given Barrier Assumptions 1-3,  $(x(\mu), u(\mu), w(\mu))$  converges to an optimal primal-dual solution as  $\mu \rightarrow 0$ .*

Because the solution to the Karush-Kuhn-Tucker conditions (8.7)-(8.10) is unique for each  $\mu > 0$ , the solution  $(x(\mu), u(\mu), w(\mu))$  traces out a path or *trajectory* in  $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n$ . Define this path by

$$\Gamma = \{(x(\mu), u(\mu), w(\mu)) \mid \mu > 0\}. \quad (8.11)$$

This path is often referred to in the literature as the *central path* or *central trajectory* associated with the linear program (*LP*). Readers familiar with homotopy theory (see Garcia and Zangwill [165]) may view  $\mu$  as a homotopy parameter. The concept of a central trajectory in linear programming is due to Sonnevend [409], Bayer and Lagarias [42] and Megiddo [324].

Solving (8.7)-(8.10) is not easy because of the nonlinear equation  $w = \mu X^{-1}e$ . In Sections 8.2 - 8.4 we develop three different path following algorithms, the primal path following, the dual path following, and the primal-dual path following algorithms. These algorithms do not follow the central trajectory  $\Gamma$  to the optimal primal-dual solution because they approximate  $w = \mu X^{-1}e$ . However, they stay “close” to the central trajectory. These three algorithms differ in the way the nonlinear equation  $w = \mu X^{-1}e$  is written and approximated. In Section 8.5 an  $O(\sqrt{n}L)$  step termination proof is provided for the primal path following algorithm. Similar polynomial termination proofs are available for dual path following and primal-dual path following algorithms. The proof shows that if we start near the central path we can stay near the central path and reduce the difference between the primal and dual objective functions at each iteration. In Section 8.6 we show that the algorithm of Karmarkar is a special case of the primal path following algorithm and that the affine scaling algorithms are a limiting case of the path following algorithms. In Section 8.7 we present the predictor-corrector method of Mehrotra [329]. The predictor-corrector method is designed to reduce the number of iterations required for the path following algorithms. Issues related to finding initial feasible solutions, treating unconstrained variables, etc. are covered in Section 8.8. Concluding remarks are in Section 8.9. Exercises are provided in Section 8.10. Because the central trajectory  $\Gamma$  is parameterized by the barrier parameter  $\mu$  we use the terms “path following” and “barrier” interchangeably.

## 8.2 PRIMAL PATH FOLLOWING

The Karush-Kuhn-Tucker conditions for  $(LP_\mu)$  are

$$A^\top u + w = c \quad (8.12)$$

$$Ax - b = 0 \quad (8.13)$$

$$w - \mu X^{-1}e = 0 \quad (8.14)$$

$$x > 0. \quad (8.15)$$

Equation (8.14) contains the term  $\mu X^{-1}e$  which is nonlinear. Rather than solve a system with a nonlinear equation, we take an approximate solution  $(x^k, u^k, w^k)$  of (8.12)-(8.15) for a given  $\mu = \mu_k$ . We assume that this approximate solution satisfies  $A^\top u^k + w^k = c$ ,  $Ax^k = b$  and  $x^k, w^k > 0$ , and use this to generate a new approximate solution  $(x^{k+1}, u^{k+1}, w^{k+1})$ . Newton's method is one way to do this. The basic idea behind Newton's method is to use a first order Taylor approximation of

$$f(x_j, w_j) = w_j - \mu_k/x_j = 0 \quad (8.16)$$

about the point  $(x_j^k, w_j^k)$  which is

$$\begin{aligned} f(x_j, w_j) &\approx f(x_j^k, w_j^k) + \nabla f(x_j^k, w_j^k)^\top \begin{bmatrix} x_j - x_j^k \\ w_j - w_j^k \end{bmatrix} \\ &= (w_j^k - \mu_k/x_j^k) + [\mu_k/(x_j^k)^2, 1] \begin{bmatrix} x_j - x_j^k \\ w_j - w_j^k \end{bmatrix}. \end{aligned}$$

Let  $\Delta x = x - x^k$ ,  $\Delta u = u - u^k$  and  $\Delta w = w - w^k$ . If  $Ax^k = b$ ,  $A^\top u^k + w^k = c$ ,  $x^k, w^k > 0$ , then the first order equations of Newton's method for (8.12)-(8.14) are

$$A^\top \Delta u + I \Delta w = 0 \quad (8.17)$$

$$A \Delta x = 0 \quad (8.18)$$

$$w^k - \mu_k X_k^{-1}e = -\mu_k X_k^{-2} \Delta x - \Delta w. \quad (8.19)$$

System (8.17)-(8.19) is an approximation of (8.12)-(8.14) since the nonlinear equation of (8.14) has been replaced by its first order approximation. Again, the reader should consult a nonlinear programming text such as Bazaraa, Sherali, and Shetty [44] or Luenberger [296] for more on Newton's method.

If  $(\Delta x, \Delta u$  and  $\Delta w)$  is a solution to (8.17)-(8.19) and  $(x^k, u^k, w^k)$  is a solution to (8.12)-(8.13) and (8.15), then for sufficiently small  $\alpha_k$ ,

$$(x^{k+1}, u^{k+1}, w^{k+1}) = (x^k, u^k, w^k) + \alpha_k (\Delta x, \Delta u, \Delta w)$$

also satisfies (8.12)-(8.13) and (8.15).

Next rewrite (8.19) as  $X_k W_k e - \mu_k e = -\mu_k X_k^{-1} \Delta x - X_k \Delta w$  where  $W_k$  is the  $n \times n$  diagonal matrix with element  $(i, i)$  equal to  $w_i^k$  and  $w^k = W_k e$ . By (8.17),

$$X_k W_k e - \mu_k e = -\mu_k X_k^{-1} \Delta x + X_k A^\top \Delta u. \quad (8.20)$$

Since  $A\Delta x = 0$ , the vector  $w = -\mu_k X_k^{-1} \Delta x$  is in the linear space  $H = \{w \mid AX_k w = 0\}$  and  $X_k A^\top \Delta u$  is an element of  $H^\perp$ . Then  $\Delta u$  which is a solution to (8.20), is also a solution to the least squares problem

$$\min_{\Delta u} \|(X_k W_k e - \mu_k e) - X_k A^\top \Delta u\|^2. \quad (8.21)$$

The solution is

$$\Delta u = (AX_k^2 A^\top)^{-1} (AX_k)(X_k W_k e - \mu_k e). \quad (8.22)$$

Using (8.20) gives

$$\begin{aligned} \Delta x &= -\frac{1}{\mu_k} X_k (X_k W_k e - \mu_k e) + \frac{1}{\mu_k} X_k (X_k A^\top) \Delta u \\ &= -\frac{1}{\mu_k} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1} (AX_k)) (X_k W_k e - \mu_k e) \end{aligned} \quad (8.23)$$

$$= -\frac{1}{\mu_k} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1} (AX_k)) (X_k c - \mu_k e) \quad (8.24)$$

Finally using (8.17) we have

$$\Delta w = -A^\top \Delta u. \quad (8.25)$$

The  $\alpha_k$  is selected so that  $w^{k+1}, x^{k+1} > 0$ . It is therefore possible to select both a “primal”  $\alpha_P^k$  so that  $x^{k+1} = x^k + \alpha_P^k \Delta x$  remains positive and a dual  $\alpha_D^k$  so that  $w^{k+1} = w^k + \alpha_D^k \Delta w$  remains positive. This is done by performing the two ratio tests

$$\alpha_P^k = \alpha \left( \min_i \{x_i^k / (-\Delta x_i) \mid \Delta x_i < 0\} \right), \quad (8.26)$$

$$\alpha_D^k = \alpha \left( \min_i \{w_i^k / (-\Delta w_i) \mid \Delta w_i < 0\} \right) \quad (8.27)$$

where  $\alpha \in (0, 1)$ . In updating  $u^{k+1}$  and  $w^{k+1}$  the same multiplier  $\alpha_D^k$  should be used on both  $\Delta u$  and  $\Delta w$  since both  $u$  and  $w$  appear in the dual constraint.

This process could be repeated several times to generate successively better solutions to (8.12)-(8.15) for a given  $\mu_k$ . However, it is more effective to simply take one Newton step at each iteration and then decrease  $\mu_k$ . This is accomplished by selecting a  $\theta \in (0, 1)$  and setting  $\mu_{k+1} = (1 - \theta)\mu_k$ . In Section 8.5 we show that for appropriate  $\mu_0, \theta$ , and  $\alpha$  this method terminates in  $O(\sqrt{n}L)$  iterations and that it is possible to find the optimal solution from this terminal interior solution in polynomial time.

One feature of the primal path following algorithm is that a feasible primal solution  $x^k$  and dual solution  $(u^k, w^k)$  is available at each iteration. By weak duality, if  $\bar{x}$  is an optimal primal solution, then  $c^\top x^k \geq c^\top \bar{x} \geq b^\top u^k$ . Therefore, if  $c^\top x^k - b^\top u^k < \epsilon$  the current primal solution  $x^k$  is an  $\epsilon$ -optimal solution in the sense that  $c^\top x^k$  cannot exceed the optimal solution value  $c^\top \bar{x}$  by more than  $\epsilon$ . The “gap”  $c^\top x^k - b^\top u^k$  can be thought of as a “duality gap” and we terminate the algorithm when this gap is sufficiently small.

#### Algorithm 8.6 (Primal Path Following Algorithm)

**Step 1: (Initialization)**  $k \leftarrow 0$ ,  $x^0, w^0 > 0$  and  $u^0$  such that  $Ax^0 = b$ ,  $A^\top u^0 + w^0 = c$ ,  $\alpha, \theta \in (0, 1)$  and  $\epsilon, \mu_0 > 0$ .

**Step 2: (Project-Least Squares)** Find the direction in which to move. Do this by solving the least squares problem

$$\min_{\Delta u} \|(X_k W_k e - \mu_k e) - X_k A^\top \Delta u\|^2 \quad (8.28)$$

and obtain the solution  $\Delta u \leftarrow (AX_k^2 A^\top)^{-1}(AX_k)(X_k W_k e - \mu_k e)$ . This defines the other directions.

$$\begin{aligned} \Delta x &\leftarrow -\frac{1}{\mu_k} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) (X_k W_k e - \mu_k e) \\ \Delta w &\leftarrow -A^\top \Delta w \end{aligned}$$

#### Step 3: (Calculate New Solution)

$$\begin{aligned} x^{k+1} &\leftarrow x^k + \alpha_P^k \Delta x \\ u^{k+1} &\leftarrow u^k + \alpha_D^k \Delta u \\ w^{k+1} &\leftarrow w^k + \alpha_D^k \Delta w \end{aligned}$$

where  $\alpha_P^k$  and  $\alpha_D^k$  are calculated from (8.26) and (8.27), respectively.

**Step 4: (Termination Test)** If  $c^\top x^k - b^\top u^k \geq \epsilon$ , update  $\mu_k \leftarrow (1 - \theta)\mu_k$ ,  $k \leftarrow k + 1$  and return to Step 2; otherwise, stop.

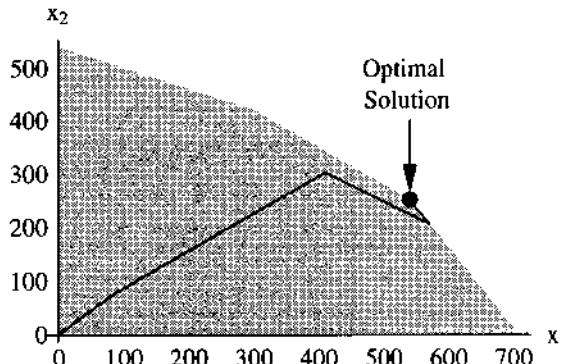
**Example 8.7** Consider Example 8.1 used in the introduction. The Mathematica implementation is given below. In Figure 8.2 we show the path of solutions for 15 iterations of the algorithm through the interior of the polytope in the solution space of the original variables  $(x_1, x_2)$ . The algorithm is converging to the optimal solution of  $(540, 252)$ . The initial point is  $x = (1, 1, 6283, 3592, 2119, 2693)^\top$ .

```

(* MATHEMATICA IMPLEMENTATION OF PRIMAL
BARRIER PATH FOLLOWING ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
alpha = .995;
n = 6;
m = 4;
theta = .5;
Iterlim = 15;
c={{-10.}, {-9.}, {0.},
 , {0.}, {0.}, {0.}};
A={{7., 10., 1.0, 0., 0., 0.},
 {3., 5., 0., 1., 0., 0.},
 {3., 2., 0., 0., 1., 0.},
 {2., 5., 0., 0., 0., 1.}};
b = { {6300.}, {3600.},
 {2124.}, {2700.} } ;
e={{1.0}, {1.0}, {1.0}, {1.0}, {1.0}, {1.0}};
x={{1.0}, {1.0}, {6283.}, {3592.},
 {2119.}, {2693.}};
u={{-1.}, {-1.}, {-1.}, {-1.}};
w={{5.}, {13.}, {1.}, {1.}, {1.}, {1.}};
X = IdentityMatrix[n];
W = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = 0.;
For[j = 1, j <= n, j++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
 For[i = 1, i <= m, i++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] ];
];
For[i = 1, i <= m, i++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[b[[i,1]]]]] ];
(* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-2.L);
(* CALCULATE INITIAL MU *)
mu = Part[Transpose[x].w, 1, 1];
dualitygap = Part[Transpose[x].w, 1, 1];
iterkount = 1;
While[( dualitygap > epsilon && iterkount <= Iterlim),
For[j = 1, j <= n, j++,
 X[[j,j]] = x[[j, 1]];
 W[[j, j]] = w[[j, 1 ]];
];
Print["x = ", x, " dualitygap =", dualitygap];
(* CALCULATE THE PRIMAL - DUAL DIRECTIONS *)
Du = Inverse[A.X.X.Transpose[A]].(A.X).(X.W.e - mu*e);
Dx = -(1./mu)*X.(IdentityMatrix[n] -

```

Figure 8.2 Solution Path From Primal Path Following Algorithm



```

(X.Transpose[A]).Inverse[A.X.X.Transpose[A]];
(A.X)).(X.W.e - mu*e);
Dw = -Transpose[A].Du;
(* PRIMAL SPACE RATIO TEST *)
alphaP = 1.0;
For[j = 1, j <= n, j++,
  alphaP = If[Dx[[j,1]] < 0,
    Min[alphaP, X[[j, j]] / -Dx[[j,1]] ],
    alphaP];
alphaP = alpha*alphaP;
(* DUAL SPACE RATIO TEST *)
alphaD = 1.%;
For[j = 1, j <= n, j++,
  alphaD = If[Dw[[j,1]] < 0,
    Min[alphaD, W[[j, j]] / -Dw[[j,1]] ],
    alphaD];
alphaD = alpha*alphaD;
(* OBTAIN THE NEW POINT *)
x = x + alphaP*Dx;
u = u + alphaD*Dw;
w = w + alphaD*Dw;
(* UPDATE MU and GAP *)
mu = (1 - theta)*mu;
dualitygap = Part[ Transpose[x].w, 1, 1 ];
iterkount = iterkount + 1;
] (* END THE WHILE STATEMENT *)

```

See Gill et al. [182] for computational results with a primal path following algorithm. Tomlin [431] compares an implementation of primal path following and simplex within WHIZARD, a state-of-the-art mathematical programming package.

### 8.3 DUAL PATH FOLLOWING

The logic behind dual path following is identical to that of primal path following. However, instead of the barrier function involving the nonnegativity primal constraints, the barrier function is formed using the nonnegativity constraints of the dual problem. The dual barrier optimization problem is

$$(DLP_\mu) \quad \begin{aligned} \max \quad & b^\top u + \mu \sum_{j=1}^n \ln(w_j) \\ \text{s.t.} \quad & A^\top u + w = c \\ & w > 0. \end{aligned}$$

The gradient of the objective function of  $(DLP_\mu)$  is  $(b - \mu W^{-1}e)^\top$ . If  $(u, w)$  is an optimal solution to  $(DLP_\mu)$  there cannot be solution to the system

$$\begin{aligned} b^\top \Delta u - \mu (W^{-1}e)^\top \Delta w &> 0 \\ A^\top \Delta u + \Delta w &= 0. \end{aligned}$$

Let  $x$  be the dual multipliers associated with the constraint  $A^\top \Delta u + \Delta w = 0$ . Using the same logic as for the primal problem  $(LP_\mu)$ , the Karush-Kuhn-Tucker optimality conditions are

$$\begin{aligned} A^\top u + w &= c \\ Ax &= b \\ \mu W^{-1}e - x &= 0 \\ w &> 0. \end{aligned}$$

The Karush-Kuhn-Tucker optimality conditions for the dual problem are equivalent to the Karush-Kuhn-Tucker optimality conditions for the primal since  $\mu W^{-1}e - x = 0$ ,  $w > 0$  is equivalent to  $w - \mu X^{-1}e = 0$ ,  $x > 0$ . As in the primal path following case, take the nonlinear function  $g(x_j, w_j) = \mu_k/w_j - x_j = 0$  and write the Newton first order approximation about the point  $(x_j^k, w_j^k)$ . The

first order approximation is

$$\begin{aligned} g(x_j, w_j) &\approx g(x_j^k, w_j^k) + \nabla g(x_j^k, w_j^k)^\top \begin{bmatrix} x_j - x_j^k \\ w_j - w_j^k \end{bmatrix} \\ &= \mu_k/w_j^k - x_j^k + \begin{bmatrix} -1, & -\mu_k/(w_j^k)^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix}. \end{aligned}$$

Using this approximation for  $g(x_j, w_j)$  the Kuhn-Tucker-Karush conditions become

$$A^\top \Delta u + \Delta w = 0 \quad (8.29)$$

$$A \Delta x = 0 \quad (8.30)$$

$$\mu_k W_k^{-1} e - x^k = \Delta x + \mu_k W_k^{-2} \Delta w. \quad (8.31)$$

Following the logic used for primal path following, we rewrite (8.31) as  $W_k X_k e - \mu_k e = -W_k \Delta x - \mu_k W_k^{-1} \Delta w$  and then from (8.29),

$$W_k X_k e - \mu_k e = -W_k \Delta x + \mu_k W_k^{-1} A^\top \Delta u. \quad (8.32)$$

Since  $A \Delta x = 0$ ,  $w = -W_k \Delta x$  is in the linear space  $H = \{w \mid AW_k^{-1}w = 0\}$  and  $\mu_k W_k^{-1} A^\top \Delta u$  is an element of  $H^\perp$ . Then  $\Delta u$  which is a solution to (8.32), is a solution to the least squares problem

$$\min_{\Delta u} \|(W_k X_k e - \mu_k e) - \mu_k W_k^{-1} A^\top \Delta u\|^2. \quad (8.33)$$

The solution is

$$\begin{aligned} \Delta u &= (AW_k^{-2}A^\top)^{-1}(AW_k^{-1})(\frac{1}{\mu_k}W_k X_k e - e) \\ &= \frac{1}{\mu_k}(AW_k^{-2}A^\top)^{-1}b - (AW_k^{-2}A^\top)^{-1}(AW_k^{-1})e. \end{aligned} \quad (8.34)$$

Combining (8.34) and (8.32) gives

$$\begin{aligned} \Delta x &= -W_k^{-1}((W_k X_k e - \mu_k e) - \mu_k W_k^{-1} A^\top \Delta u) \\ &= (-W_k^{-1} + W_k^{-2} A^\top (AW_k^{-2} A^\top)^{-1} (AW_k^{-1})) (W_k X_k e - \mu_k e). \end{aligned} \quad (8.35)$$

Finally, using (8.29)

$$\Delta w = -A^\top \Delta u.$$

#### Algorithm 8.8 (Dual Path Following Algorithm)

**Step 1: (Initialization)**  $k \leftarrow 0$ ,  $x^0, w^0 > 0$  and  $u^0$  such that  $Ax^0 = b$ ,  $A^\top u^0 + w^0 = c$ ,  $\alpha, \theta \in (0, 1)$  and  $\epsilon, \mu_0 > 0$ .

**Step 2: (Project - Least Squares)** Solve the least squares problem

$$\min_{\Delta u} \|(W_k X_k e - \mu_k e) - \mu W_k^{-1} A^\top \Delta u\|^2. \quad (8.36)$$

The solution is

$$\Delta u \leftarrow \frac{1}{\mu_k} (A W_k^{-2} A^\top)^{-1} b - (A W_k^{-2} A^\top)^{-1} (A W_k^{-1}) e$$

This defines the directions

$$\begin{aligned}\Delta x &\leftarrow \mu_k W_k^{-1} \left( (e - \frac{1}{\mu_k} W_k x^k) + W_k^{-1} A^\top \Delta u \right) \\ \Delta w &\leftarrow -A^\top \Delta u\end{aligned}$$

**Step 3: (Calculate New Solution)**

$$\begin{aligned}x^{k+1} &\leftarrow x^k + \alpha_P^k \Delta x \\ u^{k+1} &\leftarrow u^k + \alpha_D^k \Delta u \\ w^{k+1} &\leftarrow w^k + \alpha_D^k \Delta w\end{aligned}$$

where  $\alpha_P^k$  and  $\alpha_D^k$  are calculated from (8.26) and (8.27), respectively.

**Step 4: (Termination Test)** If  $c^\top x^k - b^\top u^k \geq \epsilon$ , update  $\mu_k \leftarrow (1 - \theta)\mu_k$ ,  $k \leftarrow k + 1$  and return to Step 2; otherwise, stop.

**Example 8.9** Refer to the linear program in Example 8.7. The Mathematica implementation is given below. In Figure 8.3 we show the path of solutions for 15 iterations of the algorithm through the interior of the polytope in the solution space of the original variables  $(x_1, x_2)$ . The algorithm is converging to the optimal solution of  $(540, 252)$ . The initial point is  $x = (1, 1, 6283, 3592, 2119, 2693)^\top$ .

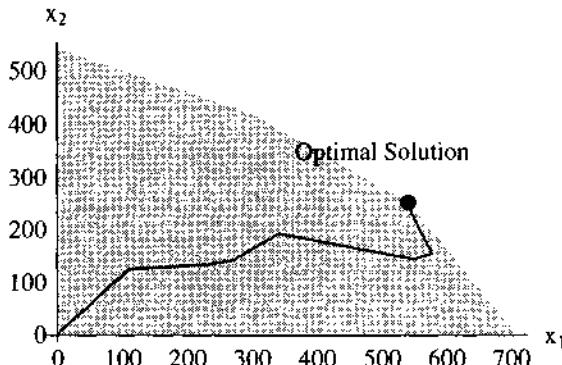
```
(* MATHEMATICA IMPLEMENTATION OF DUAL
BARRIER PATH FOLLOWING ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
alpha = .995;
n = 6;
m = 4;
theta = .5;
```

```

Iterlim = 15;
c={{-10.}, {-9.}, {0.}
, {0.}, {0.}, {0.}};
A={{7., 10., 1.0, 0., 0., 0.},
{3., 5., 0., 1., 0., 0.},
{3., 2., 0., 0., 1., 0.},
{2., 5., 0., 0., 0., 1.}};
b = {{6300.}, {3600.},
{2124.}, {2700.} };
e={{1.0}, {1.0}, {1.0}, {1.0}, {1.0}, {1.0}};
x={{1.0}, {1.0}, {6283.}, {3592.},
{2119.}, {2693.}};
u={{-1.}, {-1.}, {-1.}, {-1.}};
w={{5.}, {13.}, {1.}, {1.}, {1.}, {1.}};
X = IdentityMatrix[n];
W = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = 0.;
For[j = 1, j <= n, j++,
  L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
  For[i = 1, i <= m, i++,
    L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] - 1; - 1];
  For[i = 1, i <= m, i++,
    L = L + Ceiling[1. + Log[2, 1. + Abs[b[[i, 1]]]]] - 1];
(* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-2.L);
(* CALCULATE INITIAL MU *)
mu = Part[Transpose[x].w, 1, 1];
dualitygap = Part[Transpose[x].w, 1, 1];
iterkount = 1;
While[( dualitygap > epsilon && iterkount <= Iterlim),
  For[j = 1, j <= n, j++,
    X[[j,j]] = x[[j, 1]];
    W[[j, j]] = w[[j, 1]]; - 1;
  Print["x = ", x, " dualitygap =", dualitygap];
(* CALCULATE THE PRIMAL - DUAL DIRECTIONS *)
Du = (1./mu)*Inverse[A.Inverse[W].Inverse[W].Transpose[A]].b
- Inverse[A.Inverse[W].Inverse[W].Transpose[A]].A.Inverse[W].e;
Dx = mu*Inverse[W].((e - (1./mu)*W.x) +
Inverse[W].Transpose[A].Du);
Dw = -Transpose[A].Du;
(* PRIMAL SPACE RATIO TEST *)
alphaP = 1.0;
For[j = 1, j <= n, j++,
  alphaP = If[Dx[[j,1]] < 0,

```

**Figure 8.3** Solution Path From Dual Path Following Algorithm



```

Min[alphaP, X[[j, j]] / - Dx[[j,1]] ],
alphaP];
alphaP = alpha*alphaP;
(* DUAL SPACE RATIO TEST *)
alphaD = 1.;
For[j = 1, j <= n, j++,
alphaD = If[Dw[[j,1]] < 0,
    Min[alphaD, W[[j, j]] / - Dw[[j,1]] ],
    alphaD];
alphaD = alpha*alphaD;
(* OBTAIN THE NEW POINT *)
x = x + alphaP*Dx;
u = u + alphaD*Dw;
w = w + alphaD*Dw;
(* UPDATE MU and GAP *)
mu = (1 - theta)*mu;
dualitygap = Part[ Transpose[x].w, 1, 1 ];
iterkount = iterkount + 1;
] (* END THE WHILE STATEMENT *)

```

We are not aware of any computational studies with the dual path following algorithm.

## 8.4 PRIMAL-DUAL PATH FOLLOWING

Megiddo [324] was the first to introduce the path following methodology with the primal-dual path following method. The idea was later extended by Kojima, Mizuno, and Yoshise [271] who gave a version of the algorithm requiring  $O(nL)$  steps.

The difference between the primal and dual path following algorithms was the treatment of the Karush-Kuhn-Tucker conditions. In the primal algorithm we had the constraint  $w - \mu X^{-1}e = 0$ , which was expressed as  $\mu W^{-1}e - x = 0$  in the dual path following algorithm. While these are algebraically equivalent, they led to different formulas for the search directions. Note the different paths through the feasible region in Figures 8.2 and 8.3 respectively. A third equivalent form is  $WXe - \mu e = 0$ . This form does not correspond to any known barrier function. However, there is a very nice interpretation of the Karush-Kuhn-Tucker conditions

$$\begin{aligned} WXe &= \mu e \\ Ax &= b \\ A^T u + w &= c. \end{aligned}$$

The equations  $Ax = b$  and  $A^T u + w = c$  are primal feasibility and dual feasibility, respectively. When  $\mu = 0$ ,  $WXe = \mu e = 0$  which is complementary slackness! Thus, the Karush-Kuhn-Tucker conditions for  $\mu = 0$  are identical to the optimality conditions for linear program (*LP*). As  $\mu \rightarrow 0$  the primal-dual solutions  $(x(\mu), u(\mu), w(\mu))$  on the central trajectory  $\Gamma$  converge to a primal-dual solution satisfying the linear programming optimality conditions. Of course  $WXe = \mu e$  is nonlinear and Newton's method is still required. Let

$$h(x_j, w_j) = x_j w_j - \mu_k e = 0$$

and take the Newton first order approximation about  $(x_j^k, w_j^k)$  which is

$$\begin{aligned} h(x_j, w_j) &\approx h(x_j^k, w_j^k) + \nabla h(x_j^k, w_j^k)^T \begin{bmatrix} x_j - x_j^k \\ w_j - w_j^k \end{bmatrix} \\ &= x_j^k w_j^k - \mu_k + \begin{bmatrix} w_j^k & x_j^k \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix}. \end{aligned}$$

Using this approximation, the Karush-Kuhn-Tucker conditions are

$$A^T \Delta u + \Delta w = 0 \tag{8.37}$$

$$A \Delta x = 0 \tag{8.38}$$

$$X_k W_k e - \mu_k e = -W_k \Delta x - X_k \Delta w. \tag{8.39}$$

Unlike the primal and dual path following algorithms, there is not a least squares problem whose solution gives  $\Delta u$ . However, the system is first solved for  $\Delta u$  by substituting  $\Delta w = -A^\top \Delta u$  into (8.39). This gives

$$X_k W_k e - \mu_k e = -W_k \Delta x + X_k A^\top \Delta u.$$

Now multiply the result by  $W_k^{-1}$  to get

$$W_k^{-1}(X_k W_k e - \mu_k e) = -\Delta x + W_k^{-1} X_k A^\top \Delta u.$$

Now multiply by  $A$  and use  $A\Delta x = 0$  to get

$$AW_k^{-1}(X_k W_k e - \mu_k e) = AW_k^{-1} X_k A^\top \Delta u.$$

The solution for  $\Delta u$  is

$$\Delta u = (AW_k^{-1} X_k A^\top)^{-1} AW_k^{-1}(X_k W_k e - \mu_k e). \quad (8.40)$$

Using (8.40) and  $W_k^{-1}(X_k W_k e - \mu_k e) = -\Delta x + W_k^{-1} X_k A^\top \Delta u$  gives

$$\Delta x = -W_k^{-1} (I - X_k A^\top (AW_k^{-1} X_k A^\top)^{-1} AW_k^{-1}) (X_k W_k e - \mu_k e). \quad (8.41)$$

Finally,

$$\Delta w = -A^\top \Delta u. \quad (8.42)$$

### Algorithm 8.10 (Primal-Dual Path Following Algorithm)

**Step 1: (Initialization)**  $k \leftarrow 0$ ,  $x^0, w^0 > 0$  and  $u^0$  such that  $Ax^0 = b$ ,  $A^\top u^0 + w^0 = c$ ,  $\alpha, \theta \in (0, 1)$  and  $\epsilon, \mu_0 > 0$ .

**Step 2: (Calculate Directions)** The directions are:

$$\begin{aligned} \Delta u &\leftarrow (AW_k^{-1} X_k A^\top)^{-1} AW_k^{-1}(X_k W_k e - \mu_k e) \\ \Delta x &\leftarrow -W_k^{-1} (I - X_k A^\top (AW_k^{-1} X_k A^\top)^{-1} AW_k^{-1}) (X_k W_k e - \mu_k e) \\ \Delta w &\leftarrow -A^\top \Delta u \end{aligned}$$

**Step 3: (Calculate New Solution)**

$$\begin{aligned} x^{k+1} &\leftarrow x^k + \alpha_P^k \Delta x \\ u^{k+1} &\leftarrow u^k + \alpha_D^k \Delta u \\ w^{k+1} &\leftarrow w^k + \alpha_D^k \Delta w \end{aligned}$$

where  $\alpha_P^k$  and  $\alpha_D^k$  are calculated from (8.26) and (8.27), respectively.

**Step 4: (Termination Test)** If  $c^\top x^k - b^\top u^k \geq \epsilon$ , update  $\mu_k \leftarrow (1 - \theta)\mu_k$ ,  $k \leftarrow k + 1$  and return to Step 2; otherwise, stop.

**Example 8.11** Refer to the linear program in Example 8.7. The Mathematica implementation is given below. In Figure 8.4 we show the path of solutions for 15 iterations of the algorithm through the interior of the polytope in the solution space of the original variables  $(x_1, x_2)$ . The algorithm is converging to the optimal solution  $(540, 252)$ . The initial point is  $x = (1, 1, 6283, 3592, 2119, 2693)^\top$ .

The paths for all three barrier algorithms are shown simultaneously in Figure 8.5. It is interesting to note how different the paths are depending upon how the nonlinear equation  $WXe = \mu e$  is written. Equivalent algebraic versions (primal, dual and primal-dual) lead to distinctly different algorithms.

```
(* MATHEMATICA IMPLEMENTATION OF PRIMAL-DUAL
BARRIER PATH FOLLOWING ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
alpha = 0.995;
theta = 0.5;
n = 6;
m = 4;
Iterlim = 15;
c={{-10.}, {-9.}, {0.}
 , {0.}, {0.}, {0.}};
A={{7., 10., 1.0, 0., 0., 0.},
 {3., 5., 0., 1., 0., 0.},
 {3., 2., 0., 0., 1., 0.},
 {2., 5., 0., 0., 0., 1.}};
b = { {6300.}, {3600.},
 {2124.}, {2700.} } ;
e={{1.0}, {1.0}, {1.0}, {1.0}, {1.0}, {1.0}};
x={{1.0}, {1.0}, {6283.}, {3592.},
 {2119.}, {2693.}};
u={{-1.}, {-1.}, {-1.}, {-1.}};
w={{5.}, {13.}, {1.}, {1.}, {1.}, {1.}};
X = IdentityMatrix[n];
W = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = 0.;
For[j = 1, j <= n, j++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
 For[i = 1, i <= m, i++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] - 1; ];
 For[i = 1, i <= m, i++,
 L = L + Ceiling[1. + Log[2, 1. + Abs[b[[i,1]]]]] - 1];
 (* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-2.L);
(* CALCULATE INITIAL MU *)
```

```

mu = Part[Transpose[x].w, 1, 1];
dualitygap = Part[Transpose[x].w, 1, 1];
iterkount = 1;
While[( dualitygap > epsilon && iterkount <= Iterlim),
For[j = 1, j <=n, j++,
  X[[j,j]] = x[[j, 1]];
  W[[j, j]] = w[[j, 1]]; ];
Print["x = ", x, " dualitygap =", dualitygap];
(* CALCULATE THE PRIMAL - DUAL DIRECTIONS *)
Du = Inverse[A.Inverse[W].X.Transpose[A]].(A.Inverse[W]).(X.W.e - mu*e);
Dx = -Inverse[W].(IdentityMatrix[n] -
(X.Transpose[A]).Inverse[A.Inverse[W].X.Transpose[A]].
(A.Inverse[W])).(X.W.e - mu*e);
Dw = -Transpose[A].Du;
(* PRIMAL SPACE RATIO TEST *)
alphaP = 1.0;
For[j = 1, j <= n, j++,
  alphaP = If[Dx[[j,1]] < 0,
    Min[alphaP, X[[j, j]] / - Dx[[j,1]], alphaP]; ];
alphaP = alpha*alphaP;
(* DUAL SPACE RATIO TEST *)
alphaD = 1.;
For[j = 1, j <= n, j++,
  alphaD = If[Dw[[j,1]] < 0,
    Min[alphaD, W[[j, j]] / - Dw[[j,1]], ];
alphaD = alpha*alphaD;
(* OBTAIN THE NEW POINT *)
x = x + alphaP*Dx;
u = u + alphaD*Du;
w = w + alphaD*Dw;
(* UPDATE MU and GAP *)
mu = mu*(1.0 - theta);
dualitygap = Part[ Transpose[x].w, 1, 1 ];
iterkount = iterkount + 1;
] (* END THE WHILE STATEMENT *)

```

An important feature of the primal-dual approach is that it is possible to adjust  $\mu_k$  at each iteration in order to guarantee that the “duality gap” of  $c^T x^k - b^T u^k$  between the primal and dual solutions is reduced at each iteration. To see how this is done, let both the primal and dual step size be  $\hat{\alpha} \leq \min\{\alpha_D^k, \alpha_P^k\}$ . By weak duality, if  $(x^{k+1}, u^{k+1}, w^{k+1})$  is primal-dual feasi-

Figure 8.4 Solution Path From Primal-Dual Path Following Algorithm

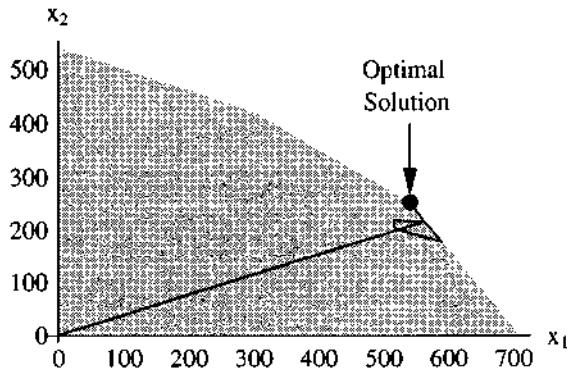
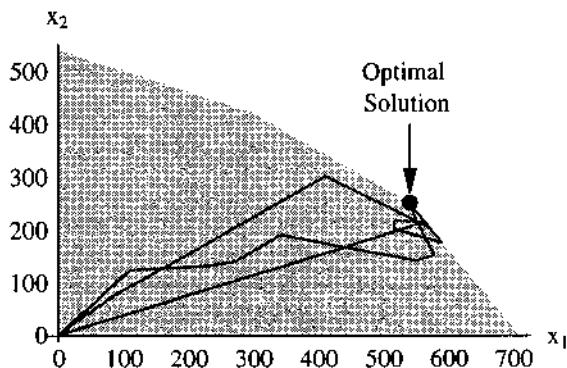


Figure 8.5 Solution Paths From Primal, Dual and Primal-Dual Path Following Algorithm



ble and  $\bar{x}$  is an optimal primal solution, then the optimal solution value  $c^\top \bar{x}$  is bounded above by  $c^\top x^{k+1}$  and below by  $b^\top u^{k+1}$ , and  $b^\top u^{k+1} \leq c\bar{x} \leq c^\top x^{k+1}$ . Let  $\epsilon_k = c^\top x^k - b^\top u^k$ . Then

$$\begin{aligned}\epsilon_k &= c^\top x^k - b^\top u^k \\ &= (A^\top u^k + w^k)^\top x^k - (Ax^k)^\top u^k \\ &= (w^k)^\top x^k.\end{aligned}$$

This gives

$$\begin{aligned}\epsilon_{k+1} &= (w^{k+1})^\top x^{k+1} \\ &= (w^k + \hat{\alpha}\Delta w)^\top (x^k + \hat{\alpha}\Delta x) \\ &= (w^k)^\top x^k + \hat{\alpha}(\Delta w)^\top x^k + \hat{\alpha}(w^k)^\top \Delta x + \hat{\alpha}^2(\Delta w)^\top \Delta x.\end{aligned}$$

By (8.17) and (8.18),  $(\Delta w)^\top = -(\Delta u)^\top A$  and  $A\Delta x = 0$ . Then  $(\Delta w)^\top \Delta x = -(\Delta u)^\top A\Delta x = 0$  and

$$\begin{aligned}\epsilon_{k+1} &= \epsilon_k + \hat{\alpha}(\Delta w)^\top x^k + \hat{\alpha}(w^k)^\top \Delta x \\ &= \epsilon_k + \hat{\alpha}((\Delta w)^\top x^k + (w^k)^\top \Delta x)\end{aligned}$$

Equation (8.39) implies  $((\Delta w)^\top x^k + (w^k)^\top \Delta x) = \mu_k n - (x^k)^\top w^k = \mu_k n - \epsilon_k$ . This gives,

$$\epsilon_{k+1} = \epsilon_k + \hat{\alpha}(\mu_k n - \epsilon_k).$$

If we modify the primal-dual path following algorithm so that the step size on both the primal and dual variables is  $\hat{\alpha}$ , then the new primal-dual solution  $(x^{k+1}, u^{k+1}, w^{k+1})$  will have a smaller duality gap than the current solution  $(x^k, u^k, w^k)$  if we select  $\mu_k$  such that  $\mu_k < (c^\top x^k - b^\top u^k)/n = \epsilon_k/n$ . This is a valid selection because  $\mu_k$  is not used in calculating  $(x^k, u^k, w^k)$ . Lustig, Marsten, and Shanno [299] and McShane, Monma, and Shanno [322] use the formula

$$\mu_k = \epsilon_k / \phi(n)$$

where  $\phi(n) = n^2$  when  $n \leq 5000$  and  $\phi(n) = n\sqrt{n}$  when  $n > 5000$ . They also use  $\alpha = 0.995$  in calculating the step size. It is the primal-dual path following algorithm that has met the most success in practice. See articles by Choi, Monma, and Shanno [90], Lustig, Marsten, and Shanno [299], Lustig, Marsten, and Shanno [302], Marsten et al. [312] and McShane, Monma, and Shanno [322]. Lustig, Marsten, and Shanno [302] computational results are given for their primal-dual code, OB1, versus the IBM simplex code OSL.

## 8.5 POLYNOMIAL TERMINATION OF PATH FOLLOWING ALGORITHMS

In the previous section three iterative algorithms were given for solving linear programs. In this section we address the termination of the algorithms and, more importantly, the polynomial termination of the algorithms. Often the proofs of termination require very specific parameter settings which are different from the ones used in practice. In this section our main concern is pedagogy. Therefore, the polynomial termination proof we present is used because it is cogent and elegantly simple. It is taken directly from Roos and Vial [385]. We give a proof of polynomial termination for a modified version of the primal path following algorithm.

Each step of the primal path following algorithm provides a primal-dual feasible pair  $(x^k, u^k, w^k)$ . By weak duality  $c^\top x^k \geq b^\top u^k$ , and the primal path following algorithm terminates when the duality gap  $c^\top x^k - b^\top u^k$  is sufficiently small, i.e. less than  $\epsilon$ . The proof we give is based on showing that at each iteration the duality gap is reduced by a factor of  $(1 - \theta)$  where  $0 < \theta < 1$ .

In all three of our path following algorithms the path we followed differed from the path of centers  $\Gamma(x(\mu), u(\mu), w(\mu))$ . In the case of the primal path following algorithm the least squares problem solved at each iteration is

$$\min_{\Delta u} \|(X_k W_k e - \mu_k e) - X_k A^\top \Delta u\|^2.$$

The dual variables are updated by  $u^{k+1} \leftarrow u^k + \alpha_D^k \Delta u$  where  $\Delta u$  is a solution to the least squares problem. Rather than update the dual variables in this fashion we modify the primal path following algorithm slightly and at iteration  $k$  solve the least squares problem

$$\min_u \|(X_k c - \mu_k e) - X_k A^\top u\|^2. \quad (8.43)$$

Denote the optimal solution to this least squares problem by  $u(x^k, \mu_k)$  and let  $u(x^k, \mu_k), w(x^k, \mu_k)$  where  $w(x^k, \mu_k) = c - A^\top u(x^k, \mu_k)$  be the vector of dual variables at iteration  $k$ . If  $\Delta w = w(x^k, \mu_k) - w^k$ , then by (8.19)

$$\Delta x = -X_k (X_k w(x^k, \mu_k)/\mu_k - e).$$

Then the solution to the least squares problem (8.43) gives the same search direction,  $\Delta x$ , in primal space used in the primal path following algorithm. This is stated formally in Lemma 8.12 which is left as an exercise.

**Lemma 8.12** If  $u(x^k, \mu_k)$  is the solution to  $\min_u \|(X_k c - \mu_k e) - X_k A^\top u\|^2$  and  $w(x^k, \mu_k) = c - A^\top u(x^k, \mu_k)$ , then

$$\begin{aligned} -X_k(X_k w(x^k, \mu_k)/\mu_k - e) &= -\frac{1}{\mu_k} X_k (I - (X_k A^\top)(A X_k A^\top)^{-1}(A X_k)) (X_k c - \mu_k e) \\ &= \Delta x. \end{aligned}$$

The reader should refer back to (8.24) to see that we are indeed using the same search direction in primal space. The motivation for rewriting the least squares problem as we have done is that we are interested in  $w(x^k, \mu_k)$ . On the central path  $\Gamma(x(\mu), u(\mu), w(\mu))$  complementary slackness holds and  $X(\mu)w(\mu)/\mu - e = 0$ . Thus, taking the Euclidean norm of  $(X_k w(x^k, \mu_k)/\mu_k - e)$  gives a measure of how far each iterate is from the central path. Observe  $\min_u \|(Xc - \mu e) - XA^\top u\|^2 = \min_{u,w} \{\|Xw - \mu e\|^2 \mid A^\top u + w = c\}$  and define distance function  $\delta(x, \mu)$  by

$$\delta(x, \mu) = \left\| \frac{Xw(x, \mu)}{\mu} - e \right\| \quad (8.44)$$

where  $w(x, \mu) = c - A^\top u(x, \mu)$  and  $u(x, \mu)$  is the solution to the least squares problem  $\min_u \|(Xc - \mu e) - XA^\top u\|^2$ . Thus, when  $x = x^k$  and  $\mu = \mu_k$ ,  $\delta(x^k, \mu_k)$  is a measure of how far the current iterate differs from the central path. If  $x = x(\mu)$ , then  $\|Xw(\mu) - \mu e\| = 0$  and  $\delta(x(\mu), \mu) = 0$ . The proof of polynomial termination rests on showing that if the primal iterate  $x^k$  is “close enough” to the central path then  $u(x^k, \mu_k)$  is dual feasible and the next primal iterate defined by  $x^{k+1} = x^k + \Delta x$  is also primal feasible.

**Lemma 8.13** If  $\delta(x^k, \mu_k) < 1$ ,  $Ax^k = b$ , and  $x^k > 0$  then  $x^{k+1} = x^k + \Delta x$  is primal feasible and  $(u(x^k, \mu_k), w(x^k, \mu_k))$ , where  $u(x^k, \mu_k)$  solves the least squares problem  $\min_u \|(X_k c - \mu e) - X_k A^\top u\|^2$ , is dual feasible.

**Proof:** By definition of  $x^{k+1}$ ,

$$\begin{aligned} x^{k+1} &= x^k + \Delta x \\ &= x^k - X_k(X_k w(x^k, \mu_k)/\mu_k - e) \quad \text{by Lemma 8.12} \\ &= 2x^k - X_k^2 w(x^k, \mu_k)/\mu_k. \end{aligned}$$

By hypothesis,  $\delta(x^k, \mu_k) < 1$ , so  $\|X_k w(x^k, \mu_k)/\mu_k - e\| < 1$  implies  $0 < x_i^k w_i(x^k, \mu_k)/\mu_k < 2$  for  $i = 1, \dots, n$ . This observation together with  $x_i^k > 0$  implies  $x_i^{k+1} = x_i^k(2 - x_i^k w_i(x^k, \mu_k)/\mu_k) > 0$  for  $i = 1, \dots, n$ . By Lemma 8.12,  $A\Delta x = 0$  which implies  $Ax^{k+1} = b$  and  $x^{k+1}$  is primal feasible.

The dual solution  $(u(x^k, \mu_k), w(x^k, \mu_k))$  is feasible if  $w(x^k, \mu_k) = c - A^\top u(x^k, \mu_k) \geq 0$ . Since  $x^k > 0$ , showing  $w(x^k, \mu_k) \geq 0$  is equivalent to showing  $X_k w(x^k, \mu_k) \geq 0$ . However, this follows immediately from the hypothesis that  $\delta(x^k, \mu_k) < 1$  since a negative component of  $X_k w(x^k, \mu_k)$  makes  $\|X_k w(x^k, \mu_k)/\mu_k - e\| > 1$ .  $\square$

Given a solution  $(x^k, u(x^k, \mu_k), w(x^k, \mu_k))$  for  $\mu_k$  that is close to the central path in the sense that  $\delta(x^k, \mu_k) < 1$ , we want to show that the closeness measure is preserved at the next iteration, that is  $\delta(x^{k+1}, \mu_{k+1}) < 1$  for a  $\mu_{k+1} < \mu_k$ . In order to do this, we first show that for a fixed  $\mu$  replacing  $x^k$  with  $x^{k+1}$  leads to a sequence of points which converges quadratically to  $x(\mu)$ .

**Lemma 8.14** *If  $\delta(x^k, \mu_k) < 1$  then  $\delta(x^{k+1}, \mu_k) \leq \delta(x^k, \mu_k)^2$ , where  $x^{k+1} = x^k + \Delta x$ .*

**Proof:** By definition,  $u^{k+1} = u(x^{k+1}, \mu)$  is the dual vector that optimizes the least squares problem  $\min_u \| (X_{k+1}c - \mu_k e) - X_{k+1}A^\top u \|^2$ , and  $w(x^{k+1}, \mu_k) = c - A^\top u(x^{k+1}, \mu_k)$ . This implies, (using  $x^{k+1} = (2x^k - X_k^2 w(x^k, \mu_k)/\mu_k)$ )

$$\begin{aligned}\delta(x^{k+1}, \mu_k)^2 &= \|X_{k+1}w(x^{k+1}, \mu_k)/\mu_k - e\|^2 \\ &= (1/\mu_k^2) \min_{u, w} \{ \|X_{k+1}w - \mu_k e\|^2 \mid A^\top u + w = c \} \\ &\leq \|X_{k+1}w(x^k, \mu_k)/\mu_k - e\|^2 \\ &= \|2X_k w(x^k, \mu_k)/\mu_k - X_k^2 W(x^k, \mu_k) w(x^k, \mu_k)/\mu_k^2 - e\|^2 \\ &= \sum_{i=1}^n (2x_i^k w_i(x^k, \mu_k)/\mu_k - (x_i^k w_i(x^k, \mu_k)/\mu_k)^2 - 1)^2 \\ &= \sum_{i=1}^n (-(x_i^k w_i(x^k, \mu_k)/\mu_k - 1)^2)^2 \\ &= \sum_{i=1}^n (x_i^k w_i(x^k, \mu_k)/\mu_k - 1)^4 \\ &\leq \left( \sum_{i=1}^n (x_i^k w_i(x^k, \mu_k)/\mu_k - 1)^2 \right)^2 \\ &= \delta(x^k, \mu_k)^4.\end{aligned}$$

Therefore,  $\delta(x^{k+1}, \mu_k) \leq \delta(x^k, \mu_k)^2$ .  $\square$

Using Lemma 8.14 we now show that, given a solution  $(x^k, u(x^k, \mu_k), w(x^k, \mu_k))$  for  $\mu_k$  which is close to the central trajectory then we can find a  $\mu_{k+1} < \mu_k$  such that  $(x^{k+1}, u(x^{k+1}, \mu_{k+1}), w(x^{k+1}, \mu_{k+1}))$  is also close to the central trajectory.

**Lemma 8.15** If  $\delta(x^k, \mu_k) \leq \frac{1}{2}$  and  $\mu_{k+1} = (1 - \theta)\mu_k$  where  $\theta = 1/(6\sqrt{n})$ , then  $\delta(x^{k+1}, \mu_{k+1}) \leq \frac{1}{2}$ .

**Proof:** By definition of  $w(x^{k+1}, \mu_{k+1})$  and  $w(x^k, \mu_k)$

$$\begin{aligned}
 \delta(x^{k+1}, \mu_{k+1}) &= ||X_{k+1}w(x^{k+1}, \mu_{k+1})/\mu_{k+1} - e|| \\
 &\leq ||X_{k+1}w(x^{k+1}, \mu_k)/\mu_{k+1} - e|| \\
 &= ||X_{k+1}w(x^{k+1}, \mu_k)/((1 - \theta)\mu_k) - e|| \\
 &= \frac{1}{(1 - \theta)} ||X_{k+1}w(x^{k+1}, \mu_k)/\mu_k - e + \theta e|| \\
 &\leq \frac{1}{(1 - \theta)} (||X_{k+1}w(x^{k+1}, \mu_k)/\mu_k - e|| + ||\theta e||) \\
 &= \frac{1}{(1 - \theta)} (\delta(x^{k+1}, \mu_k) + \theta\sqrt{n}) \\
 &\leq \frac{1}{(1 - \theta)} (\delta(x^k, \mu_k)^2 + \theta\sqrt{n}) \text{ from Lemma 8.14} \\
 &\leq \frac{1}{(1 - \theta)} (\frac{1}{4} + \frac{1}{6}) \\
 &= \frac{5}{12(1 - \theta)} \\
 &\leq \frac{1}{2}. \quad \square
 \end{aligned}$$

The significance of these lemmas is that if we successively generate  $x^{k+1} = x^k + \Delta x$  and  $\mu_{k+1} = (1 - 1/(6\sqrt{n}))\mu_k$  the duality gap between  $c^\top x^k$  and  $b^\top u(x^k, \mu_k)$  goes to zero geometrically.

**Proposition 8.16** If  $\delta(x^k, \mu_k) < 1$  and  $u(x^k, \mu_k)$  solves the least squares problem  $\min_u ||(X_k c - \mu_k e) - X_k A^\top u||^2$ , then  $(u(x^k, \mu_k), w(x^k, \mu_k))$  with  $w(x^k, \mu_k) = c - A^\top u(x^k, \mu_k)$  is dual feasible and

$$\mu_k(n - \delta(x^k, \mu_k)\sqrt{n}) \leq c^\top x^k - b^\top u(x^k, \mu_k) \leq \mu_k(n + \delta(x^k, \mu_k)\sqrt{n}). \quad (8.45)$$

**Proof:** Dual feasibility of  $(u(x^k, \mu_k), w(x^k, \mu_k))$  was shown in Lemma 8.13. Since  $x^k$  is primal feasible  $Ax^k = b$ . Then

$$\begin{aligned}
 c^\top x^k - b^\top u(x^k, \mu_k) &= c^\top x^k - (x^k)^\top A^\top u(x^k, \mu_k) \\
 &= c^\top x^k - (x^k)^\top (c - w(x^k, \mu_k)) = (x^k)^\top w(x^k, \mu_k).
 \end{aligned}$$

By Cauchy-Schwarz,

$$\delta(x^k, \mu_k)\sqrt{n} = (||X_k w(x^k, \mu_k)/\mu_k - e||)(||e||) \geq |(x^k)^\top w(x^k, \mu_k)/\mu_k - n|$$

and this implies

$$\mu_k(n - \delta(x^k, \mu_k)\sqrt{n}) \leq (x^k)^\top w(x^k, \mu_k) \leq \mu_k(n + \delta(x^k, \mu_k)\sqrt{n})$$

and the result is proved.  $\square$

#### Algorithm 8.17 (Modified Primal Path Following Algorithm)

**Step 1: (Initialization)**  $k \leftarrow 0$ ,  $x^0 > 0$  such that  $Ax^0 = b$  and  $\mu_0$  such that  $\delta(x^0, \mu_0) \leq \frac{1}{2}$ ,  $\theta \leftarrow 1/(6\sqrt{n})$  and a termination parameter  $\epsilon > 0$ .

**Step 2: (Project - Least Squares)** Find the direction in which to move. Do this by solving the least squares problem

$$\min_u ||(X_k c - \mu_k e) - X_k A^\top u||^2$$

and obtaining the dual solution  $u(x^k, \mu_k) = (AX_k^2 A^\top)^{-1}(AX_k)(X_k c - \mu_k e)$ , and  $w(x^k, \mu_k) \leftarrow (c - A^\top u(x^k, \mu_k))$ . This defines the primal direction:

$$\Delta x \leftarrow -\frac{1}{\mu} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k))(X_k c - \mu_k e).$$

#### Step 3: (Calculate New Solution)

$$x^{k+1} \leftarrow x^k + \Delta x = 2x^k - X_k^2 w(x^k, \mu_k)/\mu_k$$

**Step 4: (Termination Test)** If  $c^\top x^k - b^\top u(x^k, \mu_k) \geq \epsilon$ , update  $\mu_k \leftarrow (1 - \theta)\mu_k$ ,  $k \leftarrow k + 1$  and return to Step 2; otherwise, stop.

**Proposition 8.18** The modified primal path algorithm terminates after

$$O(\sqrt{n} \max\{\log_2(\epsilon^{-1}), \log_2(n), \log_2(\mu_0)\})$$

iterations.

**Proof:** By Proposition 8.16,

$$c^\top x^k - b^\top u(x^k, \mu_k) \leq \mu_k(n + \delta(x^k, \mu_k)\sqrt{n}) = (1 - \theta)^k \mu_0(n + \delta(x^k, \mu_k)\sqrt{n})$$

so we want to find  $k$  such that  $(1 - \theta)^k \mu_0(n + \delta(x^k, \mu_k)\sqrt{n}) < \epsilon$ . By hypothesis  $\theta = 1/(6\sqrt{n})$  and by Lemma 8.15,  $\delta(x^k, \mu_k) < \frac{1}{2}$  for all  $k$  so we need

$$\begin{aligned} (1 - \theta)^k &< \frac{\epsilon}{\mu_0(n + \frac{1}{2}\sqrt{n})} \\ k \log_2(1 - \theta) &< \log_2(\epsilon) - \log_2(\mu_0) - \log_2(n + \frac{1}{2}\sqrt{n}) \\ -k \log_2(1 - \theta) &> -\log_2(\epsilon) + \log_2(\mu_0) + \log_2(n + \frac{1}{2}\sqrt{n}). \end{aligned}$$

Since  $-\log_2(1 - \theta) > \theta$  select  $k$  such that

$$\begin{aligned} k &> (1/\theta) \left( -\log_2(\epsilon) + \log_2(\mu_0) + \log_2(n + \frac{1}{2}\sqrt{n}) \right) \\ k &> 6\sqrt{n} \left( -\log_2(\epsilon) + \log_2(\mu_0) + \log_2(n + \frac{1}{2}\sqrt{n}) \right) \end{aligned}$$

Then the number of steps is  $O(\sqrt{n} \max\{\log_2(\epsilon^{-1}), \log_2(n), \log_2(\mu_0)\})$ .  $\square$

**Example 8.19** Refer to the linear program in Example 8.7. The Mathematica implementation is given below. In Figure 8.6 we show the path of solutions for 25 iterations of the algorithm through the interior of the polytope in the solution space of the original variables  $(x_1, x_2)$ . In this case we are using the theoretical value  $\theta = 1/(6\sqrt{n})$ . In this example  $n = 6$  so  $\theta = 0.06$ . In Figure 8.2 we show the path when  $\theta = 0.5$  for the primal path algorithm. In this example we are using the starting point

$$x = (99.2509, 87.9892, 4725.35, 2862.3, 1650.27, 2061.55)^\top$$

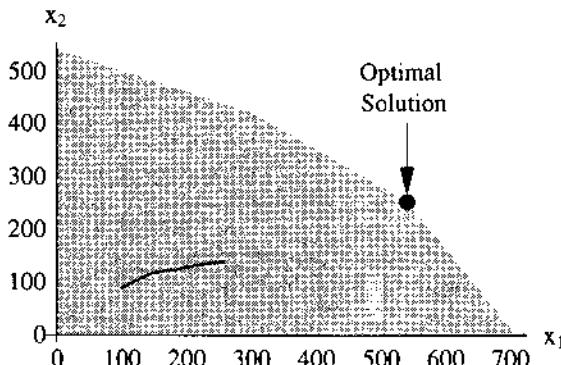
which is better than the one used in Example 8.7. The starting point used here has the property that  $\delta(x^0, \mu_0) \leq \frac{1}{2}$ . Notice how much slower convergence is for the theoretical value of  $\theta$  (in fact in Figure 8.2 the path is traced for only 15 iterations instead of 25 used in this example). The small step size is hurting convergence.

```
(* MATHEMATICA IMPLEMENTATION OF PRIMAL - DUAL
PATH FOLLOWING ALGORITHM *)
(* THE FOLLOWING IS ITERATION INDEPENDENT *)
```

```

n = 6;
m = 4;
theta = 1./(6.*N[Sqrt[n]]);
Iterlim = 25;
c={{-10.}, {-9.}, {0.}, {0.}, {0.}, {0.}};
A={{7., 10., 1.0, 0., 0., 0.},
{3., 5., 0., 1., 0.},
{3., 2., 0., 1., 0.},
{2., 5., 0., 0., 1.}};
b = {{6300.}, {3600.},
{2124.}, {2700.}};
e={{1.0}, {1.0}, {1.0}, {1.0}, {1.0}, {1.0}};
x={{99.2509}, {87.9892}, {4725.35}, {2862.3}, {1650.27},
{2061.55}};
u={{-1.}, {-1.}, {-1.}, {-1.}};
w={{5.}, {13.}, {1.}, {1.}, {1.}, {1.}};
X = IdentityMatrix[n];
W = IdentityMatrix[n];
(* CALCULATE SIZE OF INPUT DATA *)
L = 0.;
For[j = 1, j <= n, j++,
  L = L + Ceiling[1. + Log[2, 1. + Abs[c[[j, 1]]]]];
  For[i = 1, i <= m, i++,
    L = L + Ceiling[1. + Log[2, 1. + Abs[A[[i, j]]]]] - 1; - 1];
  For[i = 1, i <= m, i++,
    L = L + Ceiling[1. + Log[2, 1. + Abs[b[[i, 1]]]]] - 1];
(* CALCULATE STOPPING EPSILON *)
epsilon = 2^(-2.L);
(* CALCULATE INITIAL MU *)
mu = Part[Transpose[x].w, 1, 1];
dualitygap = Part[Transpose[x].w, 1, 1];
iterkount = 1;
While[(dualitygap > epsilon && iterkount <= Iterlim),
uold = u;
For[j = 1, j <= n, j++,
  x[[j,j]] = x[[j, 1]];
  W[[j, j]] = w[[j, 1]]; - 1;
Print["x = ", x, " dualitygap =", dualitygap];
Print["u = ", u];
(* CALCULATE THE PRIMAL - DUAL DIRECTIONS *)
u = Inverse[A.X.X.Transpose[A]].(A.X).(X.c - mu*e);
Dx = -(1./mu)*X.(IdentityMatrix[n] -
(X.Transpose[A]).Inverse[A.X.X.Transpose[A]].(A.X)).(X.W.e - mu*e);
w = c - Transpose[A].u;

```

**Figure 8.6** Solution Path From Modified Primal Path Following Algorithm

```

delta =
Part[Transpose[(X.w/mu - e)].(X.w / mu - e), 1, 1];
Print[" delta= ", delta];
(* OBTAIN THE NEW POINT *)
x = x + Dx;
(* UPDATE MU and GAP *)
mu = (1 - theta)*mu;
dualitygap = Part[ Transpose[x].w, 1, 1 ];
iterkount = iterkount + 1;
] (* END THE WHILE STATEMENT *)

```

**Lemma 8.20** If  $\hat{x}$  is a basic feasible solution for the linear program (LP) with integer data and  $(\hat{u}, \hat{w})$  is a basic feasible solution for the linear program (DLP) and  $c^T \hat{x} - b^T \hat{u} < \epsilon = 2^{-2L}$  where  $L$  is the size of linear program (LP) then  $c^T \hat{x} = b^T \hat{u}$  and  $(\hat{x}, \hat{u})$  are an optimal primal-dual solution.

**Proof:** By assumption, the input data for (LP) is integer so  $c^T \hat{x} = N_1 / \det(A_{B_1})$  where  $N_1$  is an integer and  $A_{B_1}$  is a basis matrix of  $A$ . Similarly,  $b^T \hat{u} = N_2 / \det(A_{B_2})$  where  $N_2$  is an integer and  $A_{B_2}$  is a basis matrix of  $[A^T, I_{n \times n}]$ . Then,

$$c^T \hat{x} - b^T \hat{u} = \frac{N_1}{\det(A_{B_1})} - \frac{N_2}{\det(A_{B_2})} = \frac{N_1 \det(A_{B_2}) - N_2 \det(A_{B_1})}{\det(A_{B_1}) \det(A_{B_2})}.$$

By Lemma B.4,  $\det(A_{B_1}) \leq 2^L$ . Similarly,  $\det(A_{B_2}) \leq 2^L$ . If  $c^\top \hat{x} \neq b^\top \hat{u}$  then by weak duality and the integrality of the data,

$$c^\top \hat{x} - b^\top \hat{u} \geq \frac{1}{|\det(A_{B_1})\det(A_{B_2})|} \geq \frac{1}{(2^L)(2^L)} = 2^{-2L}.$$

This contradicts the hypothesis that  $c^\top \hat{x} - b^\top \hat{u} < \epsilon = 2^{-2L}$ . Therefore,  $c^\top \hat{x} = b^\top \hat{u}$  and the optimality of  $\hat{x}$  and  $\hat{u}$  follows from strong duality.  $\square$

In the Section 8.8 we see that we can initialize the algorithm with a  $\mu_0$  which is  $O(2^L)$ . By Lemma 8.20,  $\epsilon = 2^{-2L}$ . Then by Lemma 8.18 the modified primal path algorithm terminates in  $O(\sqrt{n}L)$  iterations. Recall from Lemma 7.12 that given a feasible primal solution  $x^k$ , a primal basic feasible solution  $\hat{x}$  can be found in  $O(m^2n)$  time such that  $c^\top \hat{x} \leq c^\top x^k$ . Similarly, given the feasible dual solution  $(u^k, w^k)$  a basic feasible dual solution  $(\hat{u}, \hat{w})$  is found in  $O(m^2n)$  such that  $b^\top \hat{u} \geq b^\top u^k$ . This together with Lemma 8.20 implies that after the modified primal path algorithm terminates an optimal primal-dual solution can be found in  $O(m^2n)$  time. By the full row rank assumption on  $A$ ,  $m \leq n$ . Then each step (solving the least squares problem) of the modified primal path algorithm requires work at most  $O(n^3)$ . Therefore, the modified primal path algorithm has complexity  $O(n^{3.5}L)$ . Again, using the matrix updating methods of Karmarkar [260] and other authors such as Gonzaga [197], Monteiro and Adler [342], and Vaidya [437], this can be improved to  $O(n^3L)$ . This is the best known complexity result for linear programming. This method of Roos and Vial [385] also applies directly to the dual path method and provides termination in  $O(\sqrt{n}L)$  steps.

The first proof of an  $O(\sqrt{n}L)$  step path following algorithm for linear programming is due to Renegar [380]. Kojima, Mizuno, and Yoshise [271] showed that the primal-dual path following algorithm terminated in  $O(nL)$  steps. This was improved to  $O(\sqrt{n}L)$  by Monteiro and Adler, see [341] and [342]. See also Gonzaga [197] and Vaidya [437].

In the previous chapter the polynomiality of the Karmarkar projective algorithm was proved using a potential function. Karmarkar-like potential functions have been used by a number of other authors along with affine scaling in developing polynomial time linear programming algorithms. See, for example, Anstreicher and Bosch [14], Gonzaga [198], Mizuno, Kojima, and Ye, [340], Mizuno, Kojima, and Todd [339], Todd and Ye [427], and Ye [467]. Finally, Anstreicher [13] has the interesting result that the original barrier function method of Fiacco and McCormick [147], exactly as implemented back in 1968, solves linear and quadratic problems in  $O(\sqrt{n}L \log_2 L)$  iterations.

In the proof of polynomial termination just given, the parameter  $\mu$  is reduced by the factor of  $(1 - \theta) = (1 - 1/(6\sqrt{n}))$ . If  $n$  is say, 10000, then  $(1 - \theta) \approx 0.998$  which leads to extremely slow convergence in practice. This was illustrated in Example 8.19. Kojima, Megiddo, and Mizuno [270] show convergence of a primal-dual path following method that is much like those used in practice, in particular regarding a large step length and different step lengths in primal and dual space.

## 8.6 RELATION TO POLYHEDRAL TRANSFORMATION ALGORITHMS

The philosophy of interior point methods is to be able to take large steps by staying away from the boundary of the polytope. In the Chapter 7 this was accomplished by transforming the polytope so that the current iterate is at the “center” of the polytope. In this chapter preventing movement to the boundary of the polytope is accomplished by the addition of the barrier term  $-\mu \ln(x_j)$  for each nonnegative variable. These two distinct methods are more similar than might appear at first.

First, we relate the barrier path following algorithms to the affine scaling algorithms of the previous chapter. We show that as  $\mu \rightarrow 0$ , the search directions for the barrier methods approach the search directions for affine scaling. Then we present the results of Gill et al. [182] and show that Karmarkar’s projective algorithm is equivalent to primal path following for appropriate parameter choices.

### 8.6.1 Barrier Path Algorithms and Affine Scaling

The search direction  $\Delta x$  for primal path following is

$$\begin{aligned}\Delta x &= -\frac{1}{\mu} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) (X_k c - \mu e) \\ &= -\frac{1}{\mu} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) X_k c \\ &\quad + X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) e.\end{aligned}$$

As  $\mu \rightarrow 0$ , the component  $X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) e$  of  $\Delta x$  is dominated by the component  $-\frac{1}{\mu} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) X_k c$  and in

in the limit the search direction  $\Delta x$  becomes

$$-X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) X_k c. \quad (8.46)$$

This is identical to the search direction  $\Delta x$  for the primal affine scaling algorithm. Refer back to equation (7.53). So as  $\mu$  goes to 0, the primal path following primal direction approaches the primal affine scaling primal direction. One can think of the term  $X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) e$  as a centering term whose weight increases with  $\mu$ .

The dual path following algorithm and the primal-dual path following algorithm are also interesting in the limit as  $\mu \rightarrow 0$ . Recall that with the dual path following algorithm the search direction in dual space is

$$\Delta u = \frac{1}{\mu} (AW_k^{-2} A^\top)^{-1} b - (AW_k^{-2} A^\top)^{-1} (AW_k^{-1}) e.$$

As  $\mu \rightarrow 0$ ,  $(AW_k^{-2} A^\top)^{-1} b$  dominates the direction  $(AW_k^{-2} A^\top)^{-1} (AW_k^{-1}) e$  and effectively becomes the search direction in dual space. This is exactly the same search direction developed in the previous chapter for the dual affine scaling algorithm. Refer back to equation (7.62).

Similarly, for the primal-dual path following algorithm, the search directions in primal-dual space given by (8.40) and (8.41) become

$$\Delta u = (AW_k^{-1} X_k A^\top)^{-1} AW_k^{-1} (X_k W_k e) \quad (8.47)$$

$$\Delta x = -W_k^{-1} (I - X_k A^\top (AW_k^{-1} X_k A^\top)^{-1} AW_k^{-1}) (X_k W_k e). \quad (8.48)$$

These search directions could also have been derived from the Karush-Kuhn-Tucker equations (8.37)-(8.39) for the primal-dual algorithm using  $\mu = 0$ . This is left as an exercise. These search directions are the basis of the primal-dual affine scaling algorithm. See Monteiro, Adler, and Resende [343] or Saigal [392]. See also Jansen, Roos, and Terlaky [242] for a polynomial primal-dual affine scaling algorithm which uses search directions different from (8.47)-(8.48). In the next section we use primal-dual affine scaling directions to find an initial approximation to a nonlinear system.

### 8.6.2 Projective Transformations and Primal Path Following

In the algorithm of Karmarkar the gradient,  $f = X_k c$ , of the objective function  $f^\top y$  is projected onto the linear space  $H = \{y \mid AX_k y = 0, e^\top y = 0\}$ . The least

squares problem is

$$\min_{u,v} \|X_k c - X_k A^\top u - ev\| = \min_{u,v} \|f - X_k A^\top u - ev\|. \quad (8.49)$$

**Lemma 8.21** *The solution to the least squares problem  $\min_{u,v} \|X_k c - X_k A^\top u - ev\|$  is*

$$u_K = (AX_k^2 A^\top)^{-1} (AX_k)(X_k c), \quad v_K = c^\top x_k / n.$$

Then  $\hat{f}$ , which is the search direction in the transformed space, is the projection of  $f$  onto  $H$  and this is equal to the residual of the least squares problem. That is,  $\hat{f} = X_k c - X_k A^\top u_K - v_K e$ . The new point in the transformed space is

$$\bar{y} = e/n - (\alpha r) \left( \frac{\hat{f}}{\|\hat{f}\|} \right) \quad (8.50)$$

for some  $\alpha \in (0, 1)$ . The new iterate in the original  $x$ -space is

$$x^{k+1} = \frac{X_k \bar{y}}{e^\top X_k \bar{y}}.$$

Let  $\tilde{\alpha} = n\alpha r / \|\hat{f}\|$ . Substituting the expression for  $\bar{y}$  from (8.50) into the expression for  $x^{k+1}$  and using the definition of  $\tilde{\alpha}$  gives

$$x^{k+1} = \frac{X_k(e/n) - X_k(\tilde{\alpha}/n)\hat{f}}{e^\top X_k ((e/n) - (\tilde{\alpha}/n)\hat{f})} = \frac{X_k e - X_k \tilde{\alpha} \hat{f}}{e^\top X_k (e - \tilde{\alpha} \hat{f})}.$$

Now, use the definition of  $\hat{f}$  and fact that a feasible  $x^k$  implies  $e^\top X_k e = e^\top x^k = 1$  to rewrite the denominator as follows.

$$\begin{aligned} e^\top X_k (e - \tilde{\alpha} \hat{f}) &= e^\top X_k e - e^\top X_k \tilde{\alpha} \hat{f} \\ &= e^\top X_k e - \tilde{\alpha} e^\top X_k (X_k c - X_k A^\top u_K - v_K e) \\ &= 1 - \tilde{\alpha} ((x^k)^\top (X_k c - X_k A^\top u_K) - e^\top X_k v_K e) \\ &= 1 - \tilde{\alpha} ((x^k)^\top (X_k c - X_k A^\top u_K) - v_K). \end{aligned}$$

This gives

$$x^{k+1} = \frac{1}{(1 - \tilde{\alpha}((x^k)^\top (X_k c - X_k A^\top u_K) - v_K))} (x^k - X_k \tilde{\alpha} \hat{f})$$

**Lemma 8.22** If

$$\begin{aligned}\mu_K &= (x^k)^\top (X_k c - X_k A^\top u_K) \\ \gamma &= 1 / (1 + \tilde{\alpha}(v_K - \mu_K)) \\ \Delta x_K &= \mu_K x^k - X_k (X_k c - X_k A^\top u_K)\end{aligned}$$

then

$$x^{k+1} = \gamma \left( x^k - X_k \tilde{\alpha} \hat{f} \right) = x^k + \tilde{\alpha} \gamma \Delta x_K. \quad (8.51)$$

**Proof:**

$$\begin{aligned}x^k + \tilde{\alpha} \gamma \Delta x_K &= x^k + \frac{\tilde{\alpha} (\mu_K x^k - X_k (X_k c - X_k A^\top u_K))}{1 + \tilde{\alpha}(v_K - \mu_K)} \\ &= \frac{x^k + \tilde{\alpha} v_K x^k - x^k \tilde{\alpha} \mu_K + \tilde{\alpha} (\mu_K x^k - X_k (X_k c - X_k A^\top u_K))}{1 + \tilde{\alpha}(v_K - \mu_K)} \\ &= \frac{x^k + \tilde{\alpha} v_K x^k - \tilde{\alpha} X_k (X_k c - X_k A^\top u_K)}{1 + \tilde{\alpha}(v_K - \mu_K)} \\ &= \gamma(x^k - \tilde{\alpha} X_k \hat{f}) \quad \square\end{aligned}$$

The search direction in  $x$ -space from the Karmarkar algorithm is  $\Delta x_K$ . Following Gill et al. [182] we show that for the appropriate selection of  $\mu$ , the primal path following algorithm will generate the same search direction.

The linear program ( $LP$ ) in the standard Karmarkar form is (although there is no need to assume that the optimal objective function value is 0)

$$(KLP) \quad \begin{aligned}\min \quad & c^\top x \\ \text{s.t.} \quad & Ax = 0 \\ & e^\top x = 1 \\ & x \geq 0.\end{aligned}$$

The least squares problem for primal path following applied to ( $KLP$ ) is

$$\min_{\Delta u, \Delta v} \|(X_k W_k e - \mu e) - X_k A^\top \Delta u - X_k e \Delta v\|^2.$$

If  $(u^k, v^k, w^k)$  is a feasible dual solution, i.e.  $A^\top u^k + e v^k + w^k = c$ , and  $\Delta u = u - u^k, \Delta v = v - v^k$ , then this least squares problem can be rewritten as

$$\min_{u, v} \|(X_k c - \mu e) - X_k A^\top u - X_k e v\|^2.$$

Notice the very close similarity of this least squares problem to the least squares problem for the Karmarkar algorithm given by (8.49). They are identical except for the additional  $\mu e$  term in the primal path following least squares problem and the  $ev$  term which is now multiplied by  $X_k$ .

**Lemma 8.23** *If  $\mu = \mu_K = (x^k)^\top (X_k c - X_k A^\top u_K)$  then the optimal solution  $u_B, v_B$  to the least squares problem*

$$\min_{u,v} \|(X_k c - \mu e) - X_k A^\top u - X_k e v\|^2$$

*is  $v_B = 0$  and  $u_B = u_K = (AX_k^2 A^\top)(AX_k)(X_k c)$  and the primal path search direction  $\Delta x_B$  is*

$$\Delta x_B = \frac{1}{\mu_K} \Delta x_K. \quad (8.52)$$

**Proof:** Using the definition of Euclidean norm,  $AX_k e = Ax^k = 0$  and  $e^\top X_k e = e^\top x^k = 1$  gives

$$\begin{aligned} \|(X_k c - \mu e) - X_k A^\top u - X_k e v\|^2 &= \|X_k c - \mu e\|^2 \\ &- 2c^\top X_k^2 A^\top u + u^\top (AX_k^2 A^\top) u + 2v e^\top X_k^2 A^\top u - 2v e^\top X_k^2 c \\ &\quad + v(e^\top X_k^2 e)v + 2\mu v. \end{aligned}$$

Taking the partials with respect to  $u$  and  $v$  gives the normal equations

$$\begin{aligned} (AX_k^2 A^\top)u + AX_k^2 ev &= AX_k^2 c \\ e^\top X_k^2 ev + e^\top X_k^2 A^\top u &= e^\top X_k^2 c - \mu. \end{aligned}$$

If  $\mu = \mu_k = (x^k)^\top (X_k c - X_k A^\top u_K)$ , then the normal equation from the partial with respect to  $v$  becomes

$$\begin{aligned} e^\top X_k^2 ev + e^\top X_k^2 A^\top u &= e^\top X_k^2 c - (x^k)^\top (X_k c - X_k A^\top u_K) \\ &= e^\top X_k^2 A^\top u_K. \end{aligned}$$

It is easy to see that the unique solution to the normal equations is  $u_B = u_K$  and  $v_B = 0$ .

The Newton approximation equation for primal path following (refer back to (8.20) ) applied to the linear program (*KLP*) is

$$X_k W_k e - \mu e = -\mu X_k^{-1} \Delta x_B + X_k A^\top \Delta u + X_k e \Delta v.$$

Again, using the substitution  $\Delta u = u - u^k$  and  $\Delta v = v - v^k$  gives

$$X_k c - \mu e = -\mu X_k^{-1} \Delta x_B + X_k A^\top u + X_k \epsilon v.$$

Using the solution  $u = u_B = u_K = (AX_k^2 A^\top)^{-1}(AX_k)(X_k c)$ ,  $v_B = 0$  and solving for  $\Delta x$  gives

$$\Delta x_B = -\frac{1}{\mu} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) (X_k c - \mu e).$$

Using  $\mu = \mu_K$  and the fact that  $AX_k \mu_K e = \mu_K Ax^k = 0$  gives

$$\Delta x_B = x^k - \frac{1}{\mu_K} X_k (I - (X_k A^\top)(AX_k^2 A^\top)^{-1}(AX_k)) (X_k c) = \frac{1}{\mu_K} \Delta x_K.$$

Thus  $\Delta x_B$  differs from  $\Delta x_K$  by a factor of  $1/\mu_K$  and the lemma is proved.  $\square$

The following proposition is now immediate.

**Proposition 8.24** *If the algorithm of Karmarkar and the primal path following algorithm are applied to problem (KLP) with  $x^k$  as the current iterate, then for  $\mu = \mu_K$ , the search directions  $\Delta x_K$  and  $\Delta x_B$  are parallel and for  $\alpha_P^k = \bar{\alpha}\gamma\mu_K$ , the next iterate  $x^{k+1}$  is identical for primal path following and Karmarkar's algorithm.*

## 8.7 PREDICTOR-CORRECTOR ALGORITHMS

Predictor-corrector methods are not new to interior point methods but have their roots in the numerical solution of differential equations. See, for example, Conte and de Boor [96]. Garcia and Zangwill [165] also use predictor-corrector methods for path following algorithms. The predictor-corrector ideas applied to primal-dual path following interior point algorithms are due to Mehrotra [329]. The development below is from Mehrotra [328] and Lustig, Marsten, and Shanno [300]. In the last section relating polyhedral transformation algorithms to path following algorithms we saw that path following algorithms had both an “affine” component and a “centering” component. The predictor-corrector methods uses the current iterate  $(x^k, u^k, w^k)$  to generate an affine predictor direction. The affine predictor direction is then used 1) to estimate the centering parameter  $\mu$ , and 2) to generate a corrector step.

A primal-dual solution  $(\bar{x}, \bar{u}, \bar{w})$  is primal-dual optimal to the linear program (*LP*) and its dual (*DLP*) if and only if

$$Ax = b \quad (8.53)$$

$$A^T u + w = c \quad (8.54)$$

$$XWe = 0 \quad (8.55)$$

$$x, w \geq 0. \quad (8.56)$$

Let  $\hat{h}(x_j, w_j) = x_j w_j$ . Instead of just taking a first order approximation to  $\hat{h}(x_j, w_j)$ , take a second order approximation and write:

$$\hat{h}(x_j, w_j) \approx x_j^k w_j^k + [w_j^k, x_j^k] \begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix} + \frac{1}{2} [\Delta x, \Delta w] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \end{bmatrix}$$

The second order approximation of (8.53)-(8.56) about the point  $(x^k, u^k, w^k)$  is

$$A(x^k + \Delta x) = b \quad (8.57)$$

$$A^T(u^k + \Delta u) + (w^k + \Delta w) = c \quad (8.58)$$

$$X_k \Delta w + W_k \Delta x = -X_k W_k e - \Delta X \Delta W e \quad (8.59)$$

$$(x^k + \Delta x, w^k + \Delta w) \geq 0. \quad (8.60)$$

Equations (8.57)-(8.59) are identical to the equations used in the primal-dual path following method except that in equation (8.59) the centering term  $\mu$  is missing and there is an additional nonlinear term  $\Delta X \Delta W e$  from the second order approximation. Borrowing from the literature on the numerical solution of differential equations (see, for example, Conte and de Boor [96]), rather than solve the nonlinear system (8.57)-(8.60) the predictor-corrector method first fixes  $\Delta x \Delta W e = 0$  and solves the system

$$A(x^k + \Delta x) = b \quad (8.61)$$

$$A^T(u^k + \Delta u) + (w^k + \Delta w) = c \quad (8.62)$$

$$X_k \Delta w + W_k \Delta x = -X^k W^k e \quad (8.63)$$

for  $(\Delta \hat{x}, \Delta \hat{u}, \Delta \hat{w})$ . Since the centering term  $\mu$  was missing we refer to this direction as the affine direction. In fact, if  $(x^k, u^k, w^k)$  is primal-dual feasible, then the solution to (8.61)-(8.63) gives the primal-dual affine scaling search directions defined earlier in (8.47)-(8.48). The solution of (8.61)-(8.63) is left as an exercise. Using this solution perform the ratio tests

$$\hat{\alpha}_P = \min_j \left\{ \frac{x_j^k}{(-\Delta \hat{x}_j)} \mid \Delta \hat{x}_j < 0 \right\} \quad (8.64)$$

$$\hat{\alpha}_D = \min_j \left\{ \frac{w_j^k}{(-\Delta \hat{w}_j)} \mid \Delta \hat{x}_j < 0 \right\} \quad (8.65)$$

and then set  $\alpha_P = .99995\hat{\alpha}_P$  and  $\alpha_D = .99995\hat{\alpha}_D$ . The affine direction  $(\Delta \hat{x}, \Delta \hat{u}, \Delta \hat{w})$  vector and the step sizes  $\alpha_P, \alpha_D$  are used to estimate the complementary slackness if a step is taken in the affine direction. That is,

$$\hat{g} = (x^k + \alpha_P \Delta \hat{x})(w^k + \alpha_D \Delta \hat{w}) \quad (8.66)$$

measures the quantity by which the new iterate would miss complementary slackness if the new iterate were  $x^{k+1} = x^k + \alpha_P \Delta \hat{x}$ ,  $u^{k+1} = u^k + \alpha_D \Delta \hat{u}$ , and  $w^{k+1} = w^k + \alpha_D \Delta \hat{w}$ . Mehrotra [328] then estimates the centering parameter as

$$\mu = \left( \frac{\hat{g}}{(x^k)^\top w^k} \right)^\nu \left( \frac{(x^k)^\top w^k}{n} \right). \quad (8.67)$$

In the section on the primal-dual path following algorithm we showed that setting  $\mu = (x^k)^\top w^k / n$  results in a reduction in the duality gap at each iteration. The logic behind (8.67) is that when  $\hat{g}/(x^k)^\top w^k$  is small, good progress is made by moving in the affine direction and so the centering term should be small. However, when the affine direction results in poor progress towards reducing the complementary slackness term, more weight is given to the centering term. Experiments in Mehrotra [329] indicate that the number of primal-dual iterations is fairly stable for values of  $\nu$  between two and four. Lustig, Marsten, and Shanno [300] use (8.67) when  $(x^k)^\top w^k \geq 1$  and  $\mu = (x^k)^\top w^k / \phi(n)$  when  $(x^k)^\top w^k < 1$ .

Once the centering parameter is calculated using (8.67) the following system is solved.

$$\begin{aligned} A(x^k + \Delta x) &= b \\ A^\top(u^k + \Delta u) + (w^k + \Delta w) &= c \\ X_k \Delta w + W_k \Delta x &= -X^k W^k e - \Delta \hat{X} \Delta \hat{W} e + \mu e. \end{aligned}$$

This system does not require another Cholesky factorization since it is identical to (8.61)-(8.63) except for the constants on the right hand side. Thus, only one additional back solve is required. The solution  $(\Delta x, \Delta u, \Delta w)$  to this system is used to determine the new solution  $(x^{k+1}, u^{k+1}, w^{k+1})$ .

Computational results reported in Lustig, Marsten, and Shanno [300] and Mehrotra [329] show that using the predictor-corrector method results in a

significant reduction in the number of primal-dual iterations required. It has become an important part of barrier codes. Lustig, Marsten, and Shanno [301] adapt the methods of Kojima, Megiddo, and Mizuno [270] to provide a convergent primal-dual predictor-corrector algorithm.

The method just described involves one correction step. One could keep iterating and use multiple corrections by repeatedly solving (8.57)-(8.59) with the nonlinear term  $\Delta x \Delta w$  fixed at the current estimate, obtaining a new estimate  $(\Delta \hat{x}, \Delta \hat{u}, \Delta \hat{w})$ , and substituting it back into the right hand side of (8.57)-(8.59). Carpenter, Lustig, and Mulvey [84] investigate using multiple corrections and find that little is gained from more than one correction. Higher order methods for interior point algorithms have also been explored by Bayer and Lagarias [42] and Domich et al. [127] and implemented by Adler et al. [3]. A proof of polynomiality for several variants of predictor-corrector algorithms is in Zhang and Zhang [478].

## 8.8 OTHER ISSUES

### 8.8.1 Initial Feasibility

The primal, dual and primal-dual path following methods assume that an initial primal-dual feasible solution is available. Here we describe how to find such a solution, including an initial solution that satisfies the condition  $\delta(x^0, \mu_0) \leq (1/2)$  in order to guarantee polynomial termination of the modified primal path following algorithm.

The primal linear program (*LP*) is first modified by adding an artificial variable,  $x_{n+1}$  in order to guarantee a feasible primal solution.

$$\begin{aligned} & \min c^\top x \\ \text{s.t. } & Ax + (b - \lambda A e)x_{n+1} = b \\ & x, x_{n+1} \geq 0 \end{aligned}$$

An interior feasible solution to this linear program is  $x = \lambda e > 0$  for  $\lambda > 0$  and  $x_{n+1} = 1$ . Since  $x_{n+1}$  is an artificial variable we would like this variable to be positive if and only if the original linear program (*LP*) is infeasible. This is done by putting a sufficiently large cost on  $x_{n+1}$  in the objective function. This cost involves  $\lambda$  and the appropriate value for  $\lambda$  is given in Lemma 8.25. In a similar fashion we guarantee dual feasibility by adding the artificial dual

variable  $u_{m+1}$  and rewriting the new dual linear program as

$$\begin{aligned} & \max b^T u \\ \text{s.t. } & A^T u - (\pi e - c)u_{m+1} + w = c \\ & w, u_{m+1} \geq 0. \end{aligned}$$

A feasible solution to this modified dual problem with strictly positive slack is  $u = 0$ ,  $w = \pi e$  for  $\pi > 0$  and  $u_{m+1} = 1$ . Again, we can put a sufficiently large cost on  $u_{m+1}$  in the objective function so that the original dual linear program is infeasible if and only if  $u_{m+1} > 0$ .

Unfortunately, after this modification, the primal and dual are not a primal-dual pair. The extra artificial primal variable  $x_{n+1}$  requires an additional dual constraint (which then results in an additional primal variable) and the additional dual variable leads to an additional primal constraint (which leads to an additional dual constraint). A feasible primal-dual pair is (see Monteiro and Adler [341])

$$\begin{aligned} \min & \quad c^T x + (\pi\lambda)x_{n+1} \\ (\text{MLP}) \quad \text{s.t. } & Ax + (b - \lambda Ae)x_{n+1} = b \\ & -(\pi e - c)^T x - \pi x_{n+2} = -(\pi\lambda(n+1) - \lambda c^T e) \\ & x, x_{n+1}, x_{n+2} \geq 0 \end{aligned}$$

$$\begin{aligned} \max & \quad b^T u - (\pi\lambda(n+1) - \lambda c^T e)u_{m+1} \\ (\text{DMLP}) \quad \text{s.t. } & A^T u - (\pi e - c)u_{m+1} + w = c \\ & (b - \lambda Ae)^T u + w_{n+1} = \pi\lambda \\ & -\pi u_{m+1} + w_{n+2} = 0 \\ & w, w_{n+1}, w_{n+2} \geq 0. \end{aligned}$$

The constraint  $-\pi u_{m+1} + w_{n+2} = 0$  and nonnegativity of  $w_{n+2}$  forces  $u_{m+1}$  to be nonnegative so  $u_{m+1}$  is not listed explicitly in the nonnegativity constraints. A primal-dual feasible solution is

$$\begin{aligned} x^0 &= \lambda e, \quad x_{n+1}^0 = 1, \quad x_{n+2}^0 = \lambda \\ u^0 &= 0, \quad u_{m+1}^0 = 1 \\ w^0 &= \pi e, \quad w_{n+1}^0 = \pi\lambda, \quad w_{n+2}^0 = \pi. \end{aligned}$$

By construction, if  $(x^0, x_{n+1}^0, x_{n+2}^0) = (\lambda, \dots, \lambda, 1, \lambda)$  and  $\mu_0 = \pi\lambda$ , then for the modified linear program (*MLP*)

$$\delta((x^0, x_{n+1}^0, x_{n+2}^0), \mu_0) = \left\| \frac{X_0 w^0}{\mu_0} - e \right\| = 0$$

and the initialization criterion  $\delta((x^0, x_{n+1}^0, x_{n+2}^0), \mu_0) \leq \frac{1}{2}$  of the modified primal path algorithm is satisfied. In fact, this initial solution actually lies on the central trajectory. It is necessary to show that the modified linear program (*MLP*) is “equivalent” to the original linear program.

**Lemma 8.25** *If  $\lambda = 2^L$  and  $\pi = 2^{2L}$ , then*

$$\pi\lambda(n+1) - \lambda c^\top e > (\pi e - c)^\top \bar{x}$$

for any basic feasible solution  $\bar{x}$  of (*LP*) and

$$\pi\lambda > (b - \lambda A e)^\top \bar{u}$$

for any basic feasible solution  $(\bar{u}, \bar{w})$  of (*DLP*).

For these values of  $\lambda$  and  $\pi$  we have the following proposition which is in Monteiro and Adler [341]. Part of the proof is from Kojima, Mizuno, and Yoshise [271].

**Proposition 8.26** *There exists an optimal primal-dual solution  $(\hat{x}, \hat{x}_{n+1}, \hat{x}_{n+2}), (\hat{u}, \hat{u}_{n+1}, \hat{w}, \hat{w}_{n+1}, \hat{w}_{n+2})$  to the modified linear program (*MLP*) and its dual (*DMLP*)*

1. If  $\hat{x}_{n+1}\hat{w}_{n+2} = 0$ , then

- 1a. if  $\hat{x}_{n+1}$  and  $\hat{w}_{n+2} = 0$  then  $\hat{x}$  is an optimal solution to (*LP*) and  $(\hat{u}, \hat{w})$  is an optimal solution to (*DLP*);
- 1b. if  $\hat{x}_{n+1} \neq 0$  then (*LP*) is infeasible;
- 1c. if  $\hat{w}_{n+2} \neq 0$  then (*LP*) is unbounded.

2. If  $\hat{x}_{n+1}\hat{w}_{n+2} \neq 0$  then (*LP*) is either infeasible or unbounded.

**Proof:** By construction both (*MLP*) and (*DMLP*) are feasible. Then (*MLP*) is not unbounded and there is an optimal primal-dual solution. First prove

part 1a and assume  $\hat{x}_{n+1} = \hat{w}_{n+2} = 0$ . By hypothesis and strong duality  $c^\top \hat{x} + \pi\lambda\hat{x}_{n+1} = b^\top \hat{u} - (\pi\lambda(n+1) - \lambda c^\top e)\hat{u}_{m+1}$ . But  $\hat{x}_{n+1} = 0$  and  $\hat{w}_{n+2} = 0$ , which implies from the constraint  $-\pi u_{m+1} + w_{n+2} = 0$  that  $u_{m+1} = 0$ . Then  $c^\top \hat{x} = b^\top \hat{u}$  and  $(\hat{x}, \hat{u}, \hat{w})$  are primal-dual feasible in  $(LP)$  and  $(DLP)$ . The result follows directly from strong duality.

Now show part 1b. Do this by showing that if there is an optimal solution to  $(LP)$  then  $\hat{x}_{n+1} = 0$ . Assume  $\tilde{x}$  is an optimal solution to  $(LP)$  and  $(\tilde{u}, \tilde{w})$  is an optimal solution to  $(DLP)$ . Define  $\tilde{x}_{n+1} = 0$  and

$$\tilde{x}_{n+2} = (1/\pi)((\pi\lambda(n+1) - \lambda c^\top e) - (\pi e - c)^\top \tilde{x}).$$

By Lemma 8.25,  $\tilde{x}_{n+2} > 0$ . Then  $(\tilde{x}, \tilde{x}_{n+1}, \tilde{x}_{n+2})$  is a feasible solution to  $(MLP)$ . If  $\hat{x}_{n+1} > 0$

$$\begin{aligned} c^\top \tilde{x} + \pi\lambda\tilde{x}_{n+1} &= \tilde{u}^\top b \\ &= \tilde{u}^\top (A\hat{x} + (b - \lambda Ae)\hat{x}_{n+1}) \\ &< (c - \tilde{w})^\top \hat{x} + \pi\lambda\hat{x}_{n+1} \quad (\text{Lemma 8.25, } A^\top \tilde{u} + \tilde{w} = c, \hat{x}_{n+1} > 0) \\ &\leq c^\top \hat{x} + \pi\lambda\hat{x}_{n+1} \quad (\tilde{w}^\top \hat{x} \geq 0). \end{aligned}$$

Then  $(\hat{x}, \hat{x}_{n+1}, \hat{x}_{n+2})$  cannot be optimal if  $\hat{x}_{n+1} > 0$ . Then  $\hat{x}_{n+1} \neq 0$  implies  $(LP)$  is either unbounded or infeasible. But  $\hat{x}_{n+1}\hat{w}_{n+2} = 0$ , so  $\hat{w}_{n+2} = 0$ . This implies  $\hat{u}_{m+1} = 0$ , so the dual problem  $(DLP)$  is feasible. Therefore,  $(LP)$  is not unbounded and must be infeasible.

The proofs of parts 1c. and 2. are similar.  $\square$

**Proposition 8.27** *The modified primal path algorithm applied to  $(MLP)$  terminates after  $O(\sqrt{n}L)$  iterations.*

**Proof:** By Proposition 8.18 the modified primal path algorithm terminates after  $O(\sqrt{n} \max\{\log_2(\epsilon^{-1}), \log_2(n), \log_2(\mu_0)\})$  iterations. By definition of  $\mu_0$ ,  $O(\log_2(\mu_0)) = O(L)$ . By Lemma 8.20  $\epsilon = 2^{-2L}$  so  $\max\{\log_2(\epsilon^{-1}), \log_2(n), \log_2(\mu_0)\}$  is  $O(L)$  and we have the desired result.  $\square$

One problem with this method is that the large cost  $2^{3L}$  in the objective function of the linear program  $(MLP)$  and the dense column  $(b - \lambda Ae)$  associated with the additional variable  $x_{n+1}$  makes the Cholesky factorization more difficult. Lustig addresses this problem in [297]. The search directions of Lustig for the primal-dual path following algorithm are derived by modifying equations

(8.37)-(8.39) to

$$\begin{aligned} X_k W_k e - \mu e &= -W_k \Delta x - X_k \Delta w \\ A \Delta x &= b - Ax^k \\ A^\top \Delta u + \Delta w &= c - A^\top u^k - w^k. \end{aligned}$$

If  $(x^k, u^k, w^k)$  are primal-dual feasible then  $b - Ax^k = 0$ ,  $c - A^\top u^k - w^k = 0$  and these equations reduce to (8.37)-(8.39). The unique solution to this system of equations is

$$\begin{aligned} \Delta u &= -(AX_k W_k^{-1} A^\top)^{-1}(AW_k^{-1}(\mu e - X_k W_k e) - \\ &\quad AX_k W_k^{-1}(c - A^\top u^k - w^k) - (b - Ax^k)) \end{aligned} \quad (8.68)$$

$$\Delta w = -A^\top \Delta u + (c - A^\top u^k - w^k) \quad (8.69)$$

$$\Delta x = W_k^{-1}(\mu e - X_k W_k e) - W_k^{-1}X_k \Delta w. \quad (8.70)$$

Just as with every algorithm considered in this chapter, the main work again involves inverting a matrix of the form  $(ADA^\top)$  where  $D$  is a diagonal matrix. The computational results in Lustig, Marsten, and Shanno [299] and Lustig, Marsten, and Shanno [302] use this method rather than modifying the original linear program through the addition of artificial variables. A convergence proof for a primal-dual path following algorithm using this method is in Kojima, Megiddo, and Mizuno [270]. Their convergence proof also allows for different step lengths in primal and dual space.

### 8.8.2 Simple Upper Bounds, Variable Upper Bounds and Unrestricted Variables

In many applications simple upper bounds are present on some or all of the primal variables. Assume the linear program  $(LP_h)$  is

$$\begin{aligned} (LP_h) \quad \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x + v = h \\ & x, v \geq 0 \end{aligned}$$

where  $h$  is a vector of simple upper bounds. For the linear program  $(LP_h)$  with upper bound vector  $h$  the logarithmic barrier problem is

$$\min c^\top x - \mu \sum_{j=1}^n \ln(x_j) - \mu \sum_{j=1}^n \ln(v_j)$$

$$(LP_{\mu,h}) \quad \begin{array}{ll} & \text{s.t.} \\ & \quad Ax = b \\ & \quad x + v = h \\ & \quad x, v > 0. \end{array}$$

The Karush-Kuhn-Tucker conditions (let  $s$  be the dual multipliers on the  $x+v = h$  constraints) for  $(LP_{\mu,h})$  are

$$\begin{aligned} c - \mu X^{-1}e - A^T u + s &= 0 \\ -\mu V^{-1}e + s &= 0 \\ Ax &= b \\ x + v &= h \\ A^T u - s + w &= c \\ x, v &> 0. \end{aligned}$$

If the primal-dual path following algorithm is used, we rewrite the Karush-Kuhn-Tucker conditions as

$$\begin{aligned} WXe &= \mu e \\ VSe &= \mu e \\ Ax &= b \\ x + v &= h \\ A^T u - s + w &= c \\ x, v &> 0. \end{aligned}$$

Applying Newton's method to an iterate with primal solution  $(x^k, v^k)$  and dual solution  $(u^k, s^k, w^k)$  as done in the previous sections gives the system

$$W\Delta x + X\Delta s = \mu e - WXe \quad (8.71)$$

$$V\Delta s + S\Delta v = \mu e - VSe \quad (8.72)$$

$$A\Delta x = b - Ax^k \quad (8.73)$$

$$\Delta x + \Delta v = h - x^k - v^k \quad (8.74)$$

$$A^T \Delta u - \Delta s + \Delta w = c - A^T u^k + s^k - w^k. \quad (8.75)$$

The solution to this system is left as an exercise.

Other special structures that arise in many applications include variable upper bound constraints of the form  $x_i \leq x_k$  and special ordered constraints of the form  $\sum_{j \in K} x_{ij} = 1$ . See Hurd and Murphy [238] for a discussion treating these special structures within a primal-dual path following algorithm.

Free or unrestricted variables also arise in many applications. As with simplex, a free variable  $x_j$  can be replaced with  $x_j^+ - x_j^-$ . Since barrier methods are not moving from extreme point to extreme point there is no guarantee that  $x_j^+ x_j^- = 0$ . In the implementation of the barrier code described in Lustig, Marsten, and Shanno [302] the authors translate the variables at each iteration so that the smaller of the two is set to a constant. They report that this method works well for problems involving several hundred free variables.

### 8.8.3 Sensitivity Analysis

In Chapter 7 we discussed recovering a basic feasible solution from the solution available when an interior point algorithm terminates. It is possible to use this basic feasible solution for sensitivity analysis as discussed in Chapter 6. However, one can also use the barrier methods to obtain the optimal partition that was described in Chapter 2. See Greenberg [208] for a discussion on the use of optimal partitions in the post optimal analysis of linear programs. See also Adler and Monteiro [4] and Güler et al. [218] where the emphasis is on finding a partition with interior point algorithms.

## 8.9 CONCLUSION

Since the initial paper of Karmarkar [260], work on non-simplex methods for linear programming has exploded exponentially. The papers by Lustig, Marsten, and Shanno [302] and Marsten et al. [312] are particularly cogent and lucid and are highly recommended. Marsten et al. [312] do an excellent job of describing the barrier methods as a combination of well known nonlinear programming techniques due to Newton, Lagrange, Fiacco and McCormick. The paper of Lustig, Marsten, and Shanno [302] is a survey paper with many of the important results through 1993. The paper by Gill et al. [182] is also particularly interesting and was the first to show that the polyhedral transformation method of Karmarkar was really a barrier algorithm in disguise. Gonzaga [201] provides an excellent tutorial on the theory of path following methods. Recent linear programming books with in-depth coverage of interior point algorithms include Fang and Puthenpura [146], Roos, Terlaky, and Vial [383], and Saigal [393]. Also of interest is the paper of Güler et al. [218] which develops basic linear programming duality theory from the standpoint of interior point algorithms. In summary, some key issues include the following.

**Matrix Inversion:** In any barrier or polyhedral transformation method the majority of the work involves factoring a matrix of the form  $(ADA^\top)$  where  $D$  is a diagonal matrix. We have not addressed this issue. This is covered later in Chapter 13. In that chapter we cover inversion methods for the structure  $(ADA^\top)$  when  $A$  is sparse, present some computational results and compare barrier methods with simplex methods.

**Practice versus Theory:** Factors affecting the empirical behavior of interior point algorithms include the selection of the barrier parameter  $\mu_k$  at each iteration, the initial primal-dual solution used to start the algorithm and the step lengths in primal and dual space,  $\alpha_P$  and  $\alpha_D$  respectively. When barrier algorithms were first implemented the parameters used in implementing the algorithms were often quite different from the theoretical values. Quite naturally, researchers actually implementing the barrier algorithms were more concerned with results than with theoretically correct parameter values. One of the most exciting developments in interior point methods has been the convergence of theory and practice.

1. **Barrier Parameter Reduction:** At each iteration of a primal, dual or primal-dual algorithm, the barrier parameter  $\mu$  is reduced by a factor of  $(1 - \theta)$ . Example 8.19 with corresponding Figure 8.6 illustrated the problem of very slow convergence associated with small reductions in  $\mu$  at each iteration. This problem was quickly discovered by people implementing interior point algorithms. A useful classification of step sizes given by den Hertog [227] is

**long-step** if  $\theta$  is a constant in the open interval  $(0,1)$  independent of input data;

**medium-step** if  $\theta = \frac{\nu}{\sqrt{n}}$  where  $\nu > 0$  is an arbitrary constant (possibly large) independent of input data;

**short-step** if  $\theta = \frac{\nu}{\sqrt{n}}$  where  $\nu > 0$  is small enough so that only one Newton step is required to stay “sufficiently close” to the central path.

The modified primal algorithm uses a short-step with  $\theta = 1/(6\sqrt{n})$ . Initially, the  $O(\sqrt{n}L)$  algorithms were short-step algorithms and practitioners were forced to use values for  $\theta$  which did not guarantee convergence. However, several researchers later discovered medium and long-step algorithms with an iteration bound of  $O(nL)$  or  $O(\sqrt{n}L)$ . Examples include den Hertog, Roos, and Vial [228], Gonzaga and Todd [202], Gonzaga [199, 200], and Roos and Vial [384].

2. **Infeasible-Interior-Point Methods:** We have already discussed the implementation difficulties caused by using the linear programs

(MLP) and (DMLP). Efficient codes for primal-dual path following such as OB1 described in Lustig, Marsten, and Shanno [299] and Lustig, Marsten, and Shanno [302] do not work with artificial variables but use the search directions based on (8.68)-(8.70). Fortunately, Kojima, Megiddo [270] proved convergence of the search directions given by (8.68)-(8.70) for an arbitrary  $x > 0$  and  $w > 0$ . This is a very significant result. Research on the so called infeasible-interior-point methods is now extensive. See, for example, Mizuno, Todd, and Kojima [339] who give a primal-dual infeasible-interior-point algorithm based on potential function reduction and Lustig, Marsten, and Shanno [301] and Potra [372] for infeasible-interior-point algorithms combined with predictor-corrector search directions. Ye, Todd, and Mizuno [472] gave the first  $O(\sqrt{n}L)$ -iteration complexity infeasible-interior-point algorithm. See also the papers by Billups and Ferris [52] and Zhang [475].

3. **Primal-Dual Steps:** In the early implementations of primal-dual algorithms, researchers found that allowing for different step lengths in primal and dual space ( $\alpha_P \neq \alpha_D$ ) increased the efficiency of the algorithm. They also discovered that large steps were important, i.e.  $\alpha_P$  and  $\alpha_D$  close to one. See, for example, McShane, Monma, and Shanno [322]. The global convergence algorithm of Kojima, Megiddo, and Mizuno [270] fortunately has these properties. In the words of Lustig, Marsten, and Shanno [303]:

*We cannot overemphasize how pleasant it has become to code algorithms such as the global convergence algorithm of Kojima, Megiddo, and Mizuno that is mentioned by Todd and have them improve performance in some cases, and almost never hurt.*

The gap between theory and practice is indeed closing. We note that in order to prove the polynomiality of Kojima, Megiddo, and Mizuno [270], Mizuno [338] does assume equal step lengths in primal and dual space.

**Superlinear and Quadratic Convergence:** For all of the barrier methods studied in this chapter, appropriate parameter settings guarantee that  $\lim_{k \rightarrow \infty} (x^k)^T w^k \rightarrow 0$ . The illustrative proof that the modified primal barrier method terminates in a polynomial number of iterations is based on showing that the duality gap converges to zero because  $(x^k)^T w^k$  is reduced by a factor  $(1 - \theta)$  at each iteration with  $\theta \in (0, 1)$ . Refer to Proposition 8.16. This factor does not change at each iteration. Stronger convergence properties of path following algorithms have been studied by Mehrotra

[330], Ye et al. [471], Zhang, Tapia, and Dennis [477], and Zhang and Tapia [476]. To illustrate the types of results these authors have, the duality gap  $(x^k)^\top w^k$  of the sequence  $\{x^k, w^k\}$  converges to zero *Q-superlinearly* if

$$\lim_{k \rightarrow \infty} \frac{(x^{k+1})^\top w^{k+1}}{(x^k)^\top w^k} = 0$$

and *Q-quadratically* to zero if

$$\lim_{k \rightarrow \infty} \frac{(x^{k+1})^\top w^{k+1}}{((x^k)^\top w^k)^2} < +\infty.$$

Ye et al. [471], for example, give an interior point algorithm requiring  $O(\sqrt{n}L)$  iterations and demonstrate quadratic convergence. No nondegeneracy assumption or convergence of  $\{x^k, w^k\}$  is required.

These papers address the convergence of the duality gap, i.e. that  $(x^k)^\top w^k$  goes to zero. We have not shown in our proofs that the sequence  $\{x^k\}$  converges to an optimal primal solution. Tapia, Ye, and Zhang [420] address the problem of convergence of the iteration sequence  $\{x^k\}$  even for problems that do not have unique solutions.

**Nonlinear Programming:** The focus of this book is linear and integer linear programming. However, the reader should be aware that methods of this chapter have been extended to the field of nonlinear (in particular convex quadratic) programming. This is not surprising since the methods of this chapter arose from solving nonlinear programming problems. For a very small sample of papers see Lasdon, Plummer, and Yu [279], Mehrotra and Sun [331], Monteiro and Adler [341], Monteiro, Adler, and Resende [343], and Ye [466]. The books by Fang and Puthenpura [146], den Hertog [227] and Nesterov and Nemirovskii [356] also contain material extending interior point methods to nonlinear programming.

**State of the Art:** As of this writing, the most robust “industrial strength” interior point codes for solving very large linear programming problems are primal-dual barrier methods. Examples include CPLEX and OSL. See Lustig and Rothberg [304] for a description of a barrier primal-dual implementation in CPLEX, Lustig, Marsten, and Shanno [302] for the OB1 implementation and Forrest and Tomlin [156] for OSL. A comparison of simplex versus interior point codes appears in Chapter 13. Most of the theoretical work is either with path-following algorithms or affine scaling algorithms with projected potential functions with a barrier term.

**Further reading:** The literature on interior point methods is enormous. We refer the reader to the extensive online bibliography due to Eberhard Kranich at the University of Wuppertal. It is available over the Internet in BIBTEXformat at:

<http://netlib.org/bib/intbib.bib>

## 8.10 EXERCISES

- 8.1 Show how (8.24) follows from (8.23).
- 8.2 Prove Lemma 8.12.
- 8.3 Show that if  $\theta = 1/(6\sqrt{n})$ , then  $-\log_2(1 - \theta) > \theta$ .
- 8.4 Prove Lemma 8.2.
- 8.5 Prove Lemma 8.3.
- 8.6 Prove Lemma 8.23.
- 8.7 Find  $(\Delta x, \Delta v, \Delta u, \Delta w, \Delta s)$ , the search directions in primal-dual space for the linear program with simple upper bounds given by equations (8.71)-(8.75).
- 8.8 Prove Lemma 8.25.
- 8.9 If  $\hat{x}_{n+1}\hat{w}_{n+2} \neq 0$  in Proposition 8.26 then  $(LP)$  is either infeasible or unbounded. Explain how one could determine which is the case, i.e. unbounded or infeasible.
- 8.10 Use a  $\mu = 0$  in the Karush-Kuhn-Tucker conditions (8.37)-(8.39) for the primal-dual algorithm and derive the primal-dual affine directions (8.47)-(8.48).
- 8.11 Take the linear program

$$\begin{aligned} \min \quad & -10x_1 - 9x_2 \\ \text{s.t.} \quad & 7x_1 + 10x_2 \leq 6300 \\ & 3x_1 + 5x_2 \leq 3600 \\ & 3x_1 + 2x_2 \leq 2124 \\ & 2x_1 + 5x_2 \leq 2700 \\ & x_1, x_2 \geq 0 \end{aligned}$$

and convert it into format (*MLP*) used for the modified primal path following algorithm. What are  $L$ ,  $\pi$  and  $\lambda$ ?

8.12 If  $A$  is an  $m \times n$  matrix, show that for any basis matrix  $A_B$ , of  $[A^\top \ I]$  that  $\det(A_B) \leq 2^{\langle A \rangle}$ .

8.13 Show that the solution to the system

$$\begin{aligned} A(x^k + \Delta x) &= b \\ A^\top(u^k + \Delta u) + (w^k + \Delta w) &= c \\ X_k \Delta w + W_k \Delta x &= -X^k W^k e \end{aligned}$$

used to generate a predictor direction is

$$\begin{aligned} \Delta \hat{u} &= (AXW^{-1}A^\top)^{-1}(AXe - AXW^{-1}(c - A^\top u^k - w^k) - (b - Ax^k)) \\ \Delta \hat{w} &= -A^\top \Delta u + (c - A^\top u^k - w^k) \\ \Delta \hat{x} &= -Xe - W^{-1}X\Delta w. \end{aligned}$$

8.14 Find the solution to the system

$$\begin{aligned} A(x^k + \Delta x) &= b \\ A^\top(u^k + \Delta u) + (w^k + \Delta w) &= c \\ X_k \Delta w + W_k \Delta x &= -X^k W^k e - \Delta \hat{X} \Delta \hat{W} e + \mu e \end{aligned}$$

used to generate the corrector direction. In this system  $\Delta \hat{X}$  is the diagonal matrix with diagonal elements  $\Delta \hat{x} = -Xe - W^{-1}X\Delta w$  and  $\Delta \hat{W}$  is the diagonal matrix with diagonal elements  $\Delta \hat{w} = -A^\top \Delta u + (c - A^\top u^k - w^k)$ .

8.15 Show that the primal path algorithm and the modified primal path algorithm will yield exactly the same sequence of iterates  $(x^k, u^k, w^k)$  if: 1)  $\alpha_P = \alpha_D = 1$  in the primal path implementation, 2) the initial point  $(x^0, u^0, w^0)$  satisfies  $\delta(x^0, \mu_0) \leq \frac{1}{2}$ , and 3)  $\mu_{k+1} = (1 - \theta)\mu_k$  where  $\theta = 1/(6\sqrt{n})$ .

8.16 Take the example problem used to illustrate the path following algorithms in this chapter and perform three iterations of the primal-dual path following algorithm modified to include the predictor-corrector directions.

---

# INTEGER PROGRAMMING

## 9.1 INTRODUCTION

Several integer linear programming models were introduced in Chapter 1. Integer linear programming is a natural extension of linear programming when the decision variables represent discrete choices such as funding a project or not, opening a warehouse or not, etc. When some of the variables in a linear optimization problem are continuous and some are discrete the corresponding optimization problem is called a *mixed integer linear program*. When all of the variables are required to be integer it is an *integer linear program*. If, in addition, all of the variables are binary it is a *binary linear program*. We use the term *integer program* loosely to mean a linear optimization problem where some or all of the variables are required to be integer (perhaps binary). The general mixed integer linear program (MIP) is

$$\begin{aligned} & \min c^\top x \\ (MIP) \quad & \text{s.t. } Ax = b \\ & x \geq 0 \\ & x_j \in \mathbb{Z}, \quad j \in I. \end{aligned}$$

In Section 9.2 we illustrate the versatility of integer programming and show that the introduction of integer variables permits many new modeling possibilities. At first, we are only concerned with illustrating the power of using integer variables in modeling. In the case of integer programming, it turns out that it is important to formulate a model that is not only logically correct, but also amenable to solution. A discussion of the quality of integer programming formulations is postponed until Chapters 15 and 16. In Section 9.3 we describe a *branch-and-bound* algorithm for solving (MIP). The success of branch-and-bound depends greatly on variable and node selection and this is the topic of

Section 9.4. In Section 9.5 we generalize the basic branch-and-bound algorithm and discuss branching on hyperplanes rather than a single variable. Concluding remarks are given in Section 9.6. Exercises are provided in Section 9.7.

## 9.2 MODELING WITH INTEGER VARIABLES

**Fixed Charge Constraints:** In many scheduling and location problems there is a fixed cost or charge associated with nonzero production or locating a facility. Let  $x_t$  represent production in time  $t$  or location  $t$ . Assume that production never exceeds  $M$  and that  $f_t$  is a fixed cost associated with a nonzero value of  $x_t$ . Then the fixed charge is captured by the constraint  $x_t \leq M y_t$  with an objective function term of  $f_t y_t$  where  $y_t$  is a binary variable. See the formulation of the multi-product dynamic lot size problem in Subsection 1.3.4 in Chapter 1 for an example of these constraints involving “big  $M$ .” It is desirable to make the big  $M$  as small as possible without cutting off any feasible solutions. This is discussed at greater length in Example 15.3 in Chapter 15.

**Minimum Batch Size:** Frequently, when an investment decision is made, say investing in a mutual fund, a minimum investment is required. Similarly, in order to achieve economies of scale there often is a requirement that if an item is produced, it must be produced in a minimum batch size. Let  $x_t$  denote the production or investment quantity and let  $B$  denote the minimum batch size. Define a binary variable  $y_t$  to be 1 if  $x_t > 0$  and 0 otherwise. Like the fixed charge case put the constraint  $x_t \leq M y_t$  in the model where  $M$  is an upper limit on  $x_t$ . Unlike the fixed charge case, we also add the constraint  $B y_t \leq x_t$ . These two constraints together force  $x_t$  to take a value at least as large as  $B$  if  $x_t$  is positive. Of course  $x_t = 0, y_t = 0$  is feasible.

**Logical Restrictions:** Consider the capital budgeting model developed in Subsection 1.3.2 in Chapter 1. The binary decision variable  $x_j$  is 1 if project  $j$  is funded and 0 if not. In many cases there may be logical restrictions such as “if project  $j$  is funded, then project  $k$  must also be funded.” This condition is modeled as  $x_j \leq x_k$ . If the logical requirement is “if project  $j$  is funded, then project  $k$  must not be funded,” then the appropriate constraint is  $x_j + x_k \leq 1$ . Many other possibilities exist.

**Piecewise Linear functions:** Many real world problems involve continuous functions that are *piecewise linear*. For example, a function  $f(x)$  might be

defined as

$$f(x) = \begin{cases} c_1x + k_1, & b_1 \leq x \leq b_2 \\ c_2x + k_2, & b_2 \leq x \leq b_3 \\ \vdots & \vdots \quad \vdots \quad \vdots \quad \vdots \\ c_nx + k_n, & b_n \leq x \leq b_{n+1} \end{cases}$$

where  $c_i b_{i+1} + k_i = c_{i+1} b_{i+1} + k_{i+1}$  for  $i = 1, \dots, n - 1$ . Examples of piecewise linear functions occur when there are increasing or decreasing returns to scale, marginal costs, etc. If  $f(x)$  is also convex, i.e.  $c_i \leq c_{i+1}$  then  $f(x)$  is easily incorporated into a linear programming model by defining  $n + 1$  additional variables  $x_1, \dots, x_{n+1}$  and replacing each instance of  $f(x)$  with  $f(b_1)x_1 + f(b_2)x_2 + \dots + f(b_{n+1})x_{n+1}$  and then adding the constraints

$$x = \sum_{i=1}^{n+1} b_i x_i, \quad \sum_{i=1}^{n+1} x_i = 1, \quad x_i \geq 0, \quad i = 1, \dots, n+1.$$

**Example 9.1** Consider the problem  $\min \{f(x) + 8y \mid x + y \geq 5, y \leq 4\}$  where

$$f(x) = \begin{cases} 10x, & 0 \leq x \leq 2 \\ 12x - 4, & 2 \leq x \leq 4 \\ 15x - 16, & 4 \leq x \leq 6 \\ 17x - 28, & 6 \leq x \leq 8 \end{cases}$$

A linear programming formulation and solution for this problem is given below.

```

MIN      20 X2 + 44 X3 + 74 X4 + 108 X5 + 8 Y
SUBJECT TO
 2)  2 X2 + 4 X3 + 6 X4 + 8 X5 - X =    0
 3)  X2 + X3 + X4 + X5 + X1 =      1
 4)  Y + X >=      5
 5)  Y <=      4
END
OBJECTIVE FUNCTION VALUE
```

1) 42.00000

VARIABLE	VALUE	REDUCED COST
X2	.500000	.000000
Y	4.000000	.000000
X	1.000000	.000000
X1	.500000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-10.000000
4)	-.000000	-10.000000
5)	.000000	2.000000

Unfortunately, this modeling does not work when  $f(x)$  is not convex. Consider the following example which is a slight modification of Example 9.1.

**Example 9.2** The problem is  $\min \{f(x) + 8y \mid x + y \geq 5, y \leq 4\}$  where

$$f(x) = \begin{cases} 10x, & 0 \leq x \leq 2 \\ 8x + 4, & 2 \leq x \leq 4 \\ 6x + 12, & 4 \leq x \leq 6 \\ 4x + 24, & 6 \leq x \leq 8 \end{cases}$$

```

MIN      20 X2 + 36 X3 + 48 X4 + 56 X5 + 8 Y
SUBJECT TO
    2)  2 X2 + 4 X3 + 6 X4 + 8 X5 - X =      0
    3)  X2 + X3 + X4 + X5 + X1 =      1
    4)  Y + X >=      5
    5)  Y <=      4
END

```

OBJECTIVE FUNCTION VALUE

1) 35.00000

VARIABLE	VALUE	REDUCED COST
X5	.625000	.000000
X	5.000000	.000000
X1	.375000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-7.000000
4)	-.000000	-7.000000

The optimal solution value of  $x = 5, y = 0$  is not correct. The problem is that we are taking a convex combination of two non-adjacent break points, in this case  $x_1$  and  $x_5$ . If  $x = .375(0) + .625(8) = 5$  the actual objective function value is 42, not 35. Taking a convex combination of nonadjacent break points will underestimate a piecewise concave function.

The problem illustrated in Example 9.2 of taking nonadjacent break points is corrected by introducing binary variables. Let

$$z_i = 1, \quad b_i \leq x \leq b_{i+1}, \quad 0 \text{ otherwise} \quad (9.1)$$

$$x_i = x, \quad b_i \leq x \leq b_{i+1}, \quad 0 \text{ otherwise} \quad (9.2)$$

Every instance of  $x$  in the formulation is replaced by  $\sum_{i=1}^n x_i$ . The following constraints are added to the formulation.

$$x_i \leq b_{i+1} z_i, \quad i = 1, \dots, n \quad (9.3)$$

$$x_i \geq b_i z_i, \quad i = 1, \dots, n \quad (9.4)$$

$$\sum_{i=1}^n z_i \leq 1 \quad (9.5)$$

In the objective function  $f(x)$  is replaced by  $\sum_{i=1}^n (y_i k_i + c_i x_i)$ .

```

MIN      4 Z2 + 12 Z3 + 24 Z4 + 10 X1 + 8 X2 + 6 X3 + 4 X4 + 8 Y
SUBJECT TO
 2)   Z1 + Z2 + Z3 + Z4 =    1
 3) - 2 Z1 + X1 <=    0
 4) - 4 Z2 + X2 <=    0
 5) - 6 Z3 + X3 <=    0
 6) - 8 Z4 + X4 <=    0
 7)   X1 >=    0
 8) - 2 Z2 + X2 >=    0
 9) - 4 Z3 + X3 >=    0
10) - 6 Z4 + X4 >=    0
11)   X1 + X2 + X3 + X4 + Y >=    5
12)   Y <=    4

```

#### OBJECTIVE FUNCTION VALUE

1) 42.00000

VARIABLE	VALUE	REDUCED COST
Z1	1.000000	10.000000
X1	1.000000	0.000000
Y	4.000000	0.000000

Because of the existence of fast, robust integer linear programming software much research has gone into modeling nonlinear problems as integer linear

programs. See Example 16.6 in Chapter 16 for another way to model piecewise linear functions.

**k of n Alternatives:** The constraint  $\sum_{i=1}^{n+1} x_i = 1$  where the  $x_i$  are integer enforces the condition that exactly one  $x_i = 1$  and all others equal 0. A generalization of this case is to require exactly  $k$  of the  $n + 1$  alternative to hold. This is modeled as  $\sum_{i=1}^{n+1} x_i = k$ .

**Disjunctive Conditions:** When we write the constraints  $Ax \leq b$  for a linear or an integer program we assume that all constraints must hold simultaneously. However, there are cases where we may want only a subset of the constraints to be enforced. For example, consider a firm which can select one of  $m$  distinct manufacturing technologies. Which manufacturing technology is selected will affect production rates, resource usage, etc. If technology  $i$  is selected, then the appropriate constraints are  $A_i x \leq b^i$ . Only one set of constraints  $A_i x \leq b^i$  will hold, depending on which technology is selected. This situation is modeled by introducing a binary variable  $y_i$  for each possible technology. Variable  $y_i = 1$  if technology  $i$  is selected, 0 otherwise.

$$A_i x \geq b^i - (1 - y_i)M e, \quad i = 1, \dots, m \quad (9.6)$$

$$\sum_{i=1}^m y_i = 1 \quad (9.7)$$

$$y_i \in \{0, 1\}, \quad i = 1, \dots, m \quad (9.8)$$

In constraint (9.6)  $e$  is a vector of 1s and is conformable with the number of rows in matrix  $A_i$ . The constraint  $\sum_{i=1}^m y_i = 1$  enforces the condition that exactly one technology is selected. Constraints with this special structure are the topic of Subsection 9.5.1. Again, big  $M$  is selected large enough so that when  $y_i = 0$  and  $y_k = 1$  the constraints  $A_i x \geq b^i - M e$  are satisfied for every  $x$  that satisfies  $A_k x \geq b^k$ . This may not always be possible. This problem is discussed again in Chapter 16. If we let  $\Gamma_i = \{x \mid A_i x \leq b^i\}$  then (9.6)-(9.8) is modeling the condition that  $x$  is in the *disjunction*  $\cup_{i=1}^m \Gamma_i$ .

The modeling enhancements in this section all use 0/1 variables. However, general integer variables are also important in many applications. For example, a firm may have to pay truckload rates regardless how much cargo is loaded on the truck and an integer variable is required for the quantity of trucks sent from a depot to a customer demand location. See, for example, Eppen, Gould, and Schmidt [140].

### 9.3 BRANCH-AND-BOUND

The decision problem “given rational  $A, b$  and  $I \subseteq \{1, \dots, n\}$ , does there exist a feasible solution to  $(MIP)$ ” is  $\mathcal{NP}$ -complete. See Garey and Johnson [166] for complexity results for general, binary and mixed integer linear programming. The solution technique for  $(MIP)$  used most often in practice is *branch-and-bound*. Branch-and-bound is a philosophy for problem solution, not really a specific algorithm. A good reference for this philosophy is Geoffrion and Marsten [176]. See also Lawler and Wood [282] and Mitten [337]. The basic idea behind branch-and-bound is to take problem  $(MIP)$  and “divide” it into a set of candidate problems. The solution sets to the candidate problems should be mutually exclusive and collectively exhaustive. There are three *major* factors affecting the efficiency of this method.

1. *Selection of a problem relaxation*: instead of solving a candidate problem as an integer program we solve a *relaxation* of it. Problem

$$(PR) \quad \min\{f(x) \mid x \in \Gamma \subseteq \mathbb{R}^n\}$$

is a relaxation of problem

$$(P) \quad \min\{g(x) \mid x \in \hat{\Gamma} \subseteq \mathbb{R}^n\}$$

if and only if  $\hat{\Gamma} \subseteq \Gamma$  and  $f(x) \leq g(x)$  for all  $x \in \hat{\Gamma}$ . It follows that if  $(PR)$  is a relaxation of  $(P)$  the optimal solution value of  $(PR)$  is less than or equal to the optimal solution value of  $(P)$ . If  $(PR)$  is relaxation of  $(P)$ , then  $(P)$  is a *restriction* of  $(PR)$ . There are two important considerations here in selecting a problem relaxation. First, the problem relaxation  $(PR)$  should be relatively easy to solve since we may be forced to solve these many times. Second, the relaxed problem  $(PR)$  should be as *tight* as possible. By a *tight relaxation* we mean that the problem relaxation  $(PR)$  is a very close approximation of the original problem  $(P)$  and that the optimal solution value is equal to, or close to the optimal solution value of the original problem. This is important for bounding purposes.

2. *Problem branching/separation*: from a current candidate problem create new candidate problems. The new candidate problems are restrictions of the parent candidate problem.
3. *Problem selection*: select a problem from the candidate list. It is important to make the selection in such a fashion that the gap between the upper and lower bounds closes rapidly and that feasible solutions are found.

All of the current commercial codes available for solving problem  $(MIP)$  use some variation of the Land and Doig [276] branch-and-bound algorithm which is based upon the linear programming relaxation of  $(MIP)$ . The *linear programming relaxation*, denoted by  $(\overline{MIP})$ , of  $(MIP)$  is derived from  $(MIP)$  by deleting the integer constraints  $x_j \in \mathbb{Z}, j \in I$ .

$$(\overline{MIP}) \quad \min \{c^T x \mid Ax = b, x \geq 0\}$$

Indeed, to see that  $(\overline{MIP})$  is a relaxation of  $(MIP)$  observe that

$$\{x \mid Ax = b, x \geq 0, x_j \in \mathbb{Z}, j \in I\} \subseteq \{x \mid Ax = b, x \geq 0\}$$

and that  $c^T x$  is the objective function of both  $(MIP)$  and  $(\overline{MIP})$ . Later, in Chapter 12 we study another problem relaxation of  $(MIP)$  which is designed to yield better bounds than the linear programming relaxation.

### Algorithm 9.3 (Linear Programming Based Branch-and-Bound)

**Step 1: (Initialization)** If there is a known feasible solution  $\bar{x}$  to  $(MIP)$  set  $z_{UB} \leftarrow c^T \bar{x}$ , if there is no known feasible solution set  $z_{UB} \leftarrow \infty$ . The feasible solution  $\bar{x}$  which gives the smallest possible value for  $z_{UB}$  is known as the *incumbent* and  $z_{UB}$  is an upper bound on the optimal solution value of  $(MIP)$ . Solve  $(\overline{MIP})$ , the linear programming relaxation of  $(MIP)$ . If  $(\overline{MIP})$  is infeasible then  $(MIP)$  is infeasible. If the  $(\overline{MIP})$  is integer for all  $j \in I$ , stop with an optimal solution to  $(MIP)$ . Otherwise, add problem  $(MIP)$  to the list of candidate problems and go to Step 2.

**Step 2: (Problem Selection)** Select a candidate problem for separation from the list of candidate problems and go to Step 3. A common rule for candidate problem selection is to select the candidate problem with the smallest (assuming a minimization) linear programming relaxation value.

**Step 3: (Branching/Separation)** The candidate problem  $(CP)$  under consideration has at least one fractional integer variable.

3.a Select a fractional integer variable,  $\bar{x}_k = n_k + f_k$  for branching purposes. Here  $n_k$  is a nonnegative integer and  $f_k$  is in the open interval  $(0, 1)$ .

3.b Create two new mixed integer programs from  $(CP)$ . Create one new candidate problem by adding the constraint  $x_k \geq n_k + 1$  to the constraint set of  $(CP)$ . Create the second new candidate problem from  $(CP)$  by adding the constraint  $x_k \leq n_k$  to the constraint set of  $(CP)$ . The two new candidate problems are restrictions of the parent candidate problem since they are created by adding a constraint to the parent.

3.c Solve the linear programming relaxation of the two new candidate problems.

- 3.c.i If the linear programming relaxation is infeasible drop the newly created problem from further consideration.
- 3.c.ii If the linear programming relaxation is integer for all  $j \in I$  update the incumbent value  $z_{UB}$  and make this solution the incumbent if necessary. Drop this problem from further consideration.
- 3.c.iii If the linear programming relaxation has at least one fractional variable  $\bar{x}_j$  for  $j \in I$  and the objective function value is strictly less than  $z_{UB}$ , add this problem to the list of candidate problems.

Go to Step 4.

**Step 4: (Optimality Test)** Delete from the list of candidate problems any problem with an objective value of its relaxation which is not strictly less than  $z_{UB}$ . Stop if this list is emptied. Otherwise, go to Step 2. If the list is emptied and  $z_{UB} = \infty$  problem (MIP) is infeasible, otherwise  $z_{UB}$  is the optimal solution value of (MIP).

In Steps 3 and 4, a candidate problem is deleted when the linear programming relaxation is infeasible, or integral, or has a linear programming relaxation value larger than  $z_{UB}$ . This process of deleting candidate problems is known as *fathoming*.

In practice, for large problems it is common to terminate the branch-and-bound algorithm with an  $\epsilon$ -optimal solution. A valid upper bound is given by  $z_{UB}$ . A valid lower bound is given by the minimum linear programming relaxation value of all the candidate problems which have not been fathomed. Given an  $\epsilon > 0$ , if the difference between the upper bound and lower bound is less than  $\epsilon$ , then the incumbent solution value is within  $\epsilon$  of the optimal solution value. In general, the efficiency of this branch-and-bound algorithm is greatly affected by how closely the linear programming relaxation values of the candidate problems approximate the optimal integer solution value. The difference between the optimal value of (MIP) and the optimal of its linear programming relaxation is called the *integrality gap*. If there is a large integrality gap between (MIP) and ( $\overline{MIP}$ ) branch-and-bound may require an enormous amount of enumeration making it impractical even on a very fast computer. An integer linear program with a small integrality gap is often called *tight*. The problem of formulating integer programs with small integrality gaps is addressed later in Chapters 15 and 16. The linear programming based branch-and-bound algorithm is illustrated in the following example.

**Example 9.4**

$$\begin{aligned}
 \text{min} \quad & -x_1 - x_2 \\
 \text{s.t.} \quad & -x_1 + 2x_2 \leq 5 \\
 & 9x_1 + 4x_2 \leq 18 \\
 & 4x_1 - 2x_2 \leq 4 \\
 & x_1, x_2 \geq 0 \\
 & x_1, x_2 \in Z
 \end{aligned}$$

The optimal solution to the linear programming relaxation of this example problem is  $x_1 = .727273$ ,  $x_2 = 2.86363$  with an optimal solution value of  $-3.590903$ . A feasible integer solution is not known at this time so  $z_{UB} = \infty$ . Both decision variables are fractional. Select variable  $x_1$  as the branching variable and create two new linear programs; one linear program with the constraint  $x_1 \leq 0$  and the other with the constraint  $x_1 \geq 1.0$ .

**Linear Program 2 :**

$$\begin{aligned}
 \text{min} \quad & -x_1 - x_2 \\
 \text{s.t.} \quad & -x_1 + 2x_2 \leq 5 \\
 & 9x_1 + 4x_2 \leq 18 \\
 & 4x_1 - 2x_2 \leq 4 \\
 & x_1 \leq 0 \\
 & x_1, x_2 \geq 0
 \end{aligned}$$

The optimal solution is  $x_1 = 0$ ,  $x_2 = 2.5$  with optimal objective function value  $-2.5$ .

**Linear Program 3 :**

$$\begin{aligned}
 \text{min} \quad & -x_1 - x_2 \\
 \text{s.t.} \quad & -x_1 + 2x_2 \leq 5 \\
 & 9x_1 + 4x_2 \leq 18 \\
 & 4x_1 - 2x_2 \leq 4 \\
 & x_1 \geq 1 \\
 & x_1, x_2 \geq 0
 \end{aligned}$$

The optimal solution is  $x_1 = 1.0$ ,  $x_2 = 2.25$  with an optimal objective function value of  $-3.25$ . There are now two candidate problems. Select candidate problem 3 for further branching since it has the smallest linear programming relaxation value. Variable  $x_2$  is the only fractional variable for linear program 3 and is used for branching. Create two new linear programs by using the constraints  $x_2 \leq 2$  and  $x_2 \geq 3$ .

$$\text{min} \quad -x_1 - x_2$$

**Linear Program 4 :**

$$\begin{aligned} \text{s.t. } & -x_1 + 2x_2 \leq 5 \\ & 9x_1 + 4x_2 \leq 18 \\ & 4x_1 - 2x_2 \leq 4 \\ & x_1 \geq 1 \\ & x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The optimal solution is  $x_1 = 1.111111$ ,  $x_2 = 2$  with an optimal objective function value of  $-3.111111$ .

$$\min \quad -x_1 - x_2$$

**Linear Program 5 :**

$$\begin{aligned} \text{s.t. } & -x_1 + 2x_2 \leq 5 \\ & 9x_1 + 4x_2 \leq 18 \\ & 4x_1 - 2x_2 \leq 4 \\ & x_1 \geq 1 \\ & x_2 \geq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Linear program 5 is infeasible. There are now two candidate problems. They are candidate problems 2 and 4. Since linear program 4 has the smallest objective function value select it for branching. Create linear programs 6 and 7 by branching on variable  $x_1 = 1.111111$ .

$$\min \quad -x_1 - x_2$$

**Linear Program 6 :**

$$\begin{aligned} \text{s.t. } & -x_1 + 2x_2 \leq 5 \\ & 9x_1 + 4x_2 \leq 18 \\ & 4x_1 - 2x_2 \leq 4 \\ & x_1 \geq 1 \\ & x_2 \leq 2 \\ & x_1 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The optimal solution is  $x_1 = 1$ ,  $x_2 = 2$  with an optimal objective function value of  $-3$ . Since this solution is integer it is the new incumbent and provides the new upper bound  $z_{UB} = -3$ .

$$\min \quad -x_1 - x_2$$

**Linear Program 7 :**

$$\text{s.t. } -x_1 + 2x_2 \leq 5$$

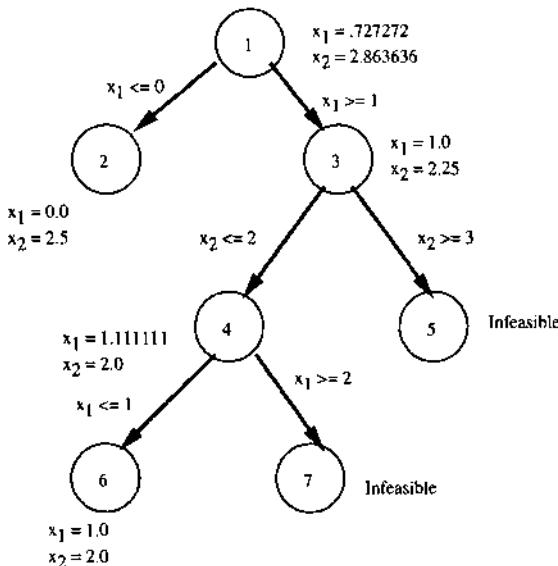
$$\begin{aligned}
 9x_1 + 4x_2 &\leq 18 \\
 4x_1 - 2x_2 &\leq 4 \\
 x_1 &\geq 1 \\
 x_2 &\leq 2 \\
 x_1 &\geq 2 \\
 x_1, x_2 &\geq 0
 \end{aligned}$$

*Linear program 7 is infeasible. Candidate problem 2 is the only candidate problem remaining in the list. The linear programming relaxation value of this candidate problem is -2.5 which is worse than the incumbent upper bound value of -3.0. Delete candidate problem 2. There are no candidate problems remaining so the optimal solution to the integer program is  $x_1 = 1, x_2 = 2$  with optimal solution value -3.*

The *branch-and-bound tree* for Example 9.4 is given in Figure 9.1. Each *node* of the tree graph represents a candidate problem. A candidate problem that has not been fathomed is also called a dangling node. We use the terms node and candidate problem interchangeably. See Moré and Wright [346] and Saltzman [395] for a survey of commercial linear programming based branch-and-bound codes and their features. Linear programming based branch-and-bound requires adding constraints of the form  $x_k \leq n_k$  or  $x_k \geq n_k + 1$  to the problem where  $n_k$  is a nonnegative integer. It is not efficient to reoptimize the modified linear program from scratch. The linear program is reoptimized using the dual simplex algorithm described in Chapter 6. The dual simplex algorithm is also used to aid in the variable and node selection. This is described in the next section.

## 9.4 NODE AND VARIABLE SELECTION

Two decisions which greatly affect the effectiveness of a linear programming based branch-and-bound algorithm are node selection in Step 2 and branching variable selection in Step 3. In their original paper, Land and Doig [276] chose the node or candidate problem with the smallest linear programming relaxation value. This will hopefully lead to a very good integer solution which helps to fathom dangling nodes. This is a breadth first strategy and can leave many dangling nodes which presents a problem of computer storage. Another node selection strategy is depth first, or last-in, first-out (LIFO) where the node selected is the most recently created node with the smallest linear programming solution value.

**Figure 9.1** Branch-and-Bound Tree

The branching variable selection is usually based upon *penalty* calculations. We first describe a method due to Beale and Small [46] and Tomlin [430]. The basic idea of the penalty method is to estimate the change in the objective function value when the constraints  $x_k \leq n_k$  or  $x_k \geq n_k + 1$  are added. This is done by calculating the change in the objective function after one-iteration of the dual simplex method. Assume fractional variable  $x_k$  has been selected for branching purposes. Write  $x_k$  as  $x_k = n_k + f_k$  where  $f_k$  is in the open interval  $(0, 1)$  and use the inequalities  $x_k \leq n_k$  and  $x_k \geq n_k + 1$  to create the two new candidate problems. Working with the linear program in standard form, this corresponds to adding the following equalities

$$x_k + s_k = n_k \quad \text{or} \quad x_k - s_k = n_k + 1, \quad \text{i.e. } s_k - x_k = -1 - n_k$$

to the linear programming relaxation. Let  $s_k$ , the slack (surplus) variable, be the basic variable for the added constraint. Since  $x_k$  is also basic, a row operation is required to maintain an identity matrix for the basic variables. Multiply row where  $x_k$  is basic by -1 and add to the newly added row. In order

to make the discussion simple, assume that  $x_k$  is basic in row  $k$ . This gives

$$\begin{aligned} s_k - \sum_{j \in N} \bar{a}_{kj} x_j &= -f_k, \quad x_k \leq n_k \text{ inequality} \\ s_k - \sum_{j \in N} \bar{a}_{kj} x_j &= (f_k - 1), \quad x_k \geq n_k + 1 \text{ inequality} \end{aligned}$$

where  $N$  is the index of nonbasic variables and  $\bar{a}_{kj}$  are updated tableau coefficients. Now perform an iteration of dual simplex on the new row of the tableau. How does this affect the current linear programming solution value at the node? If the constraint  $x_k \leq n_k$  is added, one iteration of dual simplex increases the objective function value by  $f_k(\bar{w}_q/\bar{a}_{kq})$

$$q = \operatorname{argmin} \{(\bar{w}_j/\bar{a}_{kj}) \mid \bar{a}_{kj} < 0, j \in N\}$$

is the dual simplex minimum ratio test. If the constraint  $x_k \geq n_k + 1$  is added, one iteration of dual simplex increases the objective function value by  $(1 - f_k)(\bar{w}_q/\bar{a}_{kq})$  where

$$q = \operatorname{argmin} \{-(\bar{w}_j/\bar{a}_{kj}) \mid \bar{a}_{kj} < 0, j \in N\}$$

is the dual simplex minimum ratio test. Define the penalties

$$P_k^D := f_k \left[ \frac{\bar{w}_q}{\bar{a}_{kq}} \right] \quad P_k^U := (f_k - 1) \left[ \frac{\bar{w}_q}{\bar{a}_{kq}} \right]. \quad (9.9)$$

Let  $\bar{z}_0$  denote the optimal solution value of current candidate problem linear programming relaxation. Then,  $z_0$ , the optimal integer solution value of the candidate problem must satisfy

$$z_0 \geq \bar{z}_0 + \max_{j \in I} \{ \min \{P_j^D, P_j^U\} \mid f_j > 0\}.$$

These penalties suggest several strategies.

**Strategy One:** Assume a minimization problem with  $z_{UB}$  the value of the current incumbent solution. If  $z_{UB} \leq \bar{z}_0 + \max_{j \in I} \{ \min \{P_j^D, P_j^U\} \mid f_j > 0\}$  the candidate problem is deleted from the list because it cannot produce a better optimal solution. If it is possible to produce a better integer solution, select as the branching variable the one with the smallest value of  $\min \{P_j^D, P_j^U\}$ . This branching rule will hopefully create a path to a "good" integer solution. The problem with this branching strategy is that numerous  $\bar{w}_j$  may be zero, leaving the possibility of many "ties" for variable selection and another rule must be used.

**Strategy Two:** The other extreme. If the candidate problem cannot be fathomed from the penalty calculations, select as the branching variable, the variable with the largest value of  $\min \{P_j^D, P_j^U\}$ . Then two new candidate problems

are created and the one opposite the one with the highest penalty is used for branching. The logic here is to leave a dangling node which is easily fathomed by a good incumbent.

Although branching variable selection based on the penalties in (9.9) has been successful in some instances (e.g., Little et al. [290]), it can lead to some very poor results. If a problem has several hundred or several thousand constraints, adding a new row at each node may require many dual simplex iterations. Test problems used by Forrest, Hirst, and Tomlin [154], with 2000-3000 rows required several hundred iteration of dual simplex. Since the penalty calculations are based on only one iteration of the dual simplex method, these penalties may not even be close in their prediction of objective function degradation. Forrest, Hirst, and Tomlin [154], state that "choosing the branching variables on the basis of misleading penalties can thus lead to an almost arbitrary tree search, with very little hope of attaining optimality, or even near-optimality in a reasonable time." Although improvements to these penalty calculation exist, see for example, Armstrong and Sinha [17], other criteria for variable selection in large integer programming problems are needed. A number of innovations for node and branching variable selection are suggested by Forrest, Hirst, and Tomlin [154] in their UMPIRE mixed integer programming code. See also Gauthier and G. Ribière [171] for methods used in IBM's MPSX/370-MIP/370 software. UMPIRE and MPSX/370-MIP/370 were two of the early robust commercial codes for solving mixed integer programming models.

One improvement on the basic penalty calculation used by Forrest, Hirst, and Tomlin is that of *pseudo-costs*. Let  $\bar{z}_0^j$  denote the optimal linear programming relaxation value at node  $j$ , assume variable  $k$  is selected for branching and branching on this variable generates nodes  $j + 1$  and  $j + 2$ . Define

$$P_k^U := \frac{\bar{z}^{j+1} - \bar{z}^j}{f_k}, \quad P_k^D := \frac{\bar{z}^{j+2} - \bar{z}^j}{1 - f_k} \quad (9.10)$$

The penalties defined in (9.10) can be used for branching and node selection in much the same way as the penalties defined in equation 9.9. Recently<sup>1</sup> Bixby has described a method which is a compromise of (9.9) and (9.10). He suggests calculating the penalties based on 50 dual simplex iterations and then select the branching variable from a list of 10 "good" fractional variables using the rule

$$\max_j \{10 \min \{P_k^L, P_k^U\} + \max \{P_k^L, P_k^U\}\}.$$

Finally, some commercial codes allow for the user to input priorities for the variables which reflect the importance of the integer variable to which they

---

<sup>1</sup>CSTS Conference, Dallas, Texas, January 7, 1996

are attached. The problem now becomes one of establishing these priorities. These can be established by user knowledge of the physical problem, i.e., which facilities will most affect the overall system. If there is no such knowledge a crude method would be to assign priorities in the order of the cost coefficients in the original objective function. The branching variable used is then the fractional variable with the highest priority.

## 9.5 MORE GENERAL BRANCHING

In the branch-and-bound algorithm developed in Section 9.3 candidate problems were created by adding the constraints  $x_k \leq n_k$  or  $x_k \geq n_k + 1$ . In this section we show that using more general constraints to create the subproblems can lead to considerably less enumeration. Consider the following example.<sup>2</sup>

### Example 9.5

$$\begin{aligned} & \min x_0 \\ \text{s.t. } & x_0 + 2 \sum_{j=1}^{2n} x_j = n \\ & x_0, x_j \in \{0, 1\}, \quad j = 1, \dots, 2n \end{aligned}$$

The branch-and-bound algorithm developed in Section 9.3 works very poorly with the integer program in Example 9.5. If  $n$  is an odd integer, the linear programming relaxation will have  $\lfloor n/2 \rfloor$   $x_j$  variables equal to 1.0 and one  $x_j$  variable equal to 1/2. Because all of the  $x_j$  variables have the same constraint coefficient, branching on the variable equal to 1/2 will simply force another variable to 1/2 with no effect on the objective function. One must be at least  $\lfloor n/2 \rfloor + 1$  levels down in the tree before any nodes are fathomed. However, it is valid to create two new subproblems or branches using the constraints  $\sum_{j=1}^{2n} x_j \leq \lfloor n/2 \rfloor$  and  $\sum_{j=1}^{2n} x_j \geq \lfloor n/2 \rfloor + 1$ . This partitions the set of feasible solutions to Example 9.5 into two disjoint subsets, with the union of the subsets equal to the set of feasible solutions. The constraint  $\sum_{j=1}^{2n} x_j \geq \lfloor n/2 \rfloor + 1$  results in a candidate problem which is fathomed by infeasibility and the constraint  $\sum_{j=1}^{2n} x_j \leq \lfloor n/2 \rfloor$  results in a subproblem where the linear programming relaxation solution is integer. This example illustrates the potential of generalizing the simple branching scheme of  $x_k \leq n_k$  and  $x_k \geq n_k + 1$ . If  $e^k$  is the  $k$ th

---

<sup>2</sup>This example was given to the author by J. Lee who credits H.E. Scarf.

unit vector the simple branching scheme developed in Section 9.3 corresponds to  $(e^k)^T x \leq n_k$  and  $(e^k)^T x \geq n_k + 1$ . In the following three subsections we show that there are important problem structures where replacing  $e^k$  by a more general integer vector  $v$  leads to more efficient enumeration. Since  $v^T x = v_0$  is a hyperplane we can think of the new branching methods as *branching on hyperplanes*.

### 9.5.1 Special Ordered Set Branching

Consider a problem with the following structure.

$$\min \quad \sum_{i=1}^p \sum_{j=1}^{n_i} c_{ij} x_{ij} \quad (9.11)$$

$$(MIPSOS) \quad \text{s.t.} \quad \sum_{i=1}^m \sum_{j=1}^{n_i} a_{ij}^k x_{ij} = b_k, \quad k = 1, 2, \dots, m \quad (9.12)$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \dots, p \quad (9.13)$$

$$x_{ij} \in \{0, 1\}, \quad j = 1, \dots, n_i, \quad i = 1, \dots, p \quad (9.14)$$

The  $m$  constraints (9.12) are coupling constraints and the  $p$  constraints (9.13) are generalized upper bound constraints. See Section 6.4 in Chapter 6. When the  $x_{ij}$  are required to be integer, generalized upper bound constraints are also called *type 1 special ordered set constraints* or sometimes SOS 1 constraints or just SOS constraints. See Beale and Tomlin [47]. See also Tomlin [430].

All the variables in (MIPSOS) are grouped into  $p$  subsets which are mutually disjoint and collectively exhaustive. Constraints (9.13) and (9.14), taken together, imply exactly one variable in each set is equal to one and all the rest equal zero. For this reason they are also called *multiple choice constraints*. Problems with special ordered sets of type 1 arise frequently in production scheduling models (Lasdon [277], Lasdon and Terjung [278], Pritsker, Waters, and Wolfe [373], and Sweeney and Murphy [419]), menu planning ([33]), catalog space planning (Johnson, Zoltners, and Sinha [254]), capital budgeting (Lorie and Savage [293]) and real estate (Bean, Noon, and Salton [48]). The generalized assignment problem also has this structure and has been used by Fisher and Jaikumar [149] for routing trucks, by Zoltners and Sinha [479] for sales force allocation, by Ross and Soland [388] for facility location, and by Campbell and Langevin [83] for snow removal routing. This problem structure is also

very important in the work of Sweeney and Murphy [419] which is studied in Chapter 12.

For non-trivial problems, solving the linear relaxation of (*MIPSOS*) results in an optimal solution containing fractional variables. These fractional variables are used for branching purposes in a linear programming based branch-and-bound algorithm. Consider the following two strategies.

**Strategy One: Single Variable Branching:** Select a fractional variable,  $x_{lj}$ , and branch on it. Create two descendant nodes by setting  $x_{lj} = 0$  at one node and  $x_{lj} = 1$  at the other node. This method of branching was described earlier in the chapter.

**Strategy Two: Balanced Branching:** Select an SOS constraint  $\sum_{j=1}^{n_l} x_{lj} = 1$  with fractional variables for branching. Create two new candidate problems by setting  $\sum_{j=1}^{\lfloor \frac{n_l}{2} \rfloor} x_{lj} = 0$  for one new candidate and  $\sum_{j=\lfloor \frac{n_l}{2} \rfloor + 1}^{n_l} x_{lj} = 0$  for the other candidate problem. Assume without loss of generality that the variables are ordered such that each new descendant contains at least one of the fractional variables in the linear programming relaxation at its parent.

The second strategy can be viewed as a balancing technique, or a divide and conquer strategy. Balancing techniques are often used in computer science for the design of efficient algorithms. Mergesort is an example of an algorithm based on the balancing principal (see Aho, Hopcroft, and Ullman [6] or Horowitz and Sahni [236]). Both of these strategies consist of setting a subset of variables to zero at a given node, but it is shown below that strategy two will usually lead to the implicit enumeration of more solutions. By implicitly enumerating a solution we mean eliminating that solution from further consideration without actually evaluating it. With the first branching strategy one descendant node is created by setting  $x_{lj} = 0$  and the other descendant node is created by setting  $x_{lk} = 0$ ,  $k \neq j$ . The restriction at the second node admits fewer solutions because more variables are fixed to zero. If SOS  $l$  contains  $n_l$  variables the second descendant node has  $n_l - 1$  variables fixed to zero while only one variable is fixed to zero at the first. Thus, there is a greater chance of having an infeasible problem, a linear programming relaxation value larger than the incumbent integer solution, or an all integer solution at the second node.

Define  $k_i$  = number of free integer variables (not fixed to 0/1) in SOS  $i$ ,  $i = 1, 2, \dots, p$  at node  $h$ . If  $S_h$  represents the number of solutions yet to be enumerated at node  $h$  then  $S_h = \prod_{i=1}^p k_i$ . Assume without loss, that the special order set  $\sum_{j=1}^{n_l} x_{lj} = 1$  is selected for branching and that none of the  $x_{lj}$  are fixed at

0 or 1. Since

$$\min \left\{ \frac{(n_l - \lfloor \frac{n_l}{2} \rfloor)S_h}{n_l}, \frac{\lfloor \frac{n_l}{2} \rfloor S_h}{n_l} \right\} > \min \left\{ \frac{(n_l - 1)S_h}{n_l}, \frac{S_h}{n_l} \right\} = \frac{S_h}{n_l}$$

for  $n_l \geq 4$ , strategy two enumerates more solutions with each fathom than does strategy one, unless the first strategy gets “lucky” with a fathom at node  $h + 1$ . This isn’t likely, since as previously pointed out, with strategy one a fathom is more likely at node  $h + 2$  due to the greater number of variables fixed to zero at this node.

The SOS branching scheme first implemented in MPSX/370–MIP/370 and UMPIRE (see Forrest, Hirst, and Tomlin [154]) can be viewed as compromise between strategies one and two. Rather than branching on an individual variable, or splitting the set into two equal parts, a weight  $W_{ij}$  is assigned to each SOS selected for branching purposes. Assume the variables within the special ordered set are numbered in decreasing order of their weights, i.e.,  $W_{ij_1} \geq W_{ij_2}$  if  $j_1 < j_2$ . Calculate  $\bar{W} = \sum_{j=1}^{n_l} W_{ij} \bar{x}_{ij}$  and create two new candidate problems. One candidate has  $x_{i1} + x_{i2} + \dots + x_{ir} = 0$  where  $r$  is the first index such that  $\bar{W} \geq W_{i,r+1}$  (see Gauthier and Ribière [171]). The other candidate then has  $x_{i,r+1} + x_{i,r+2} + \dots + x_{i,n_l} = 0$ . If several variables have equal weights, this procedure must be modified slightly to guarantee there is at least one non-zero variable in each subset,  $(1, r)$  and  $(r+1, n_l)$  of indices. The weights assigned by default when using MPSX/370–MIP/370 are  $W_{i1} = n_i - 1$ ,  $W_{i2} = n_i - 2, \dots$ ,  $W_{in_i} = 0$ . This is somewhat arbitrary and the objective function or a row in the constraint matrix can also be used as a “reference row” for calculating SOS weights. The survey article by Saltzman [395] lists commercial codes which have SOS branching capability.

### 9.5.2 Set Partitioning Branching

In Subsection 1.3.6 in Chapter 1 we introduced the crew scheduling problem. The crew scheduling problem is an example of a more general structure called the *set partitioning problem*. The set partitioning problem has the following structure.

$$(SP) \quad \begin{aligned} & \min c^T x \\ & \text{s.t. } Ax = e \\ & \quad x_j \in \{0, 1\}, \quad j = 1, \dots, n \end{aligned}$$

In  $(SP)$ ,  $A$  is an  $m \times n$  matrix where every element is 0 or 1 and  $e$  is an  $m$  vector of 1s. In general, the linear programming relaxation of  $(SP)$  will have

fractional  $x_j$  variables. Proposition 9.6 and the branching scheme based on it are due to Ryan and Foster [389]. The proof of Proposition 9.6 is left as an exercise.

**Proposition 9.6** *If  $\bar{x}$  is a fractional basic feasible solution of the system  $Ax = e$ ,  $x \geq 0$ , then there are index sets  $I_r := \{j \mid a_{rj} = 1\}$  and  $I_s := \{j \mid a_{sj} = 1\}$  such that*

$$0 < \sum_{j \in I_r \cap I_s} \bar{x}_j < 1. \quad (9.15)$$

Since  $\sum_{j \in I_r \cap I_s} \bar{x}_j$  is in the open interval  $(0,1)$  two candidate subproblems can be created where the subproblem for the left branch has the added constraint  $\sum_{j \in I_r \cap I_s} x_j = 1$  and the right branch has the added constraint  $\sum_{j \in I_r \cap I_s} x_j = 0$ . The advantage of this branching scheme over the single fractional variable branching scheme originally developed is that this method eliminates large numbers of solutions for both candidate problems. For the candidate problem with the added restriction  $\sum_{j \in I_r \cap I_s} x_j = 1$  only columns with  $a_{rj} = a_{sj}$  are permitted and all others are deleted. After deleting all the columns where  $a_{rj} \neq a_{sj}$  rows  $r$  and  $s$  become identical and one of them can be deleted. For the candidate problem with the added restriction  $\sum_{j \in I_r \cap I_s} x_j = 0$  the columns with  $a_{rj} = 1a_{sj}$  are deleted. In this subproblem rows  $r$  and  $s$  are disjoint (the inner product of their row matrix coefficients is 0) which increases the likelihood of integer solutions.

This branching method has proved effective in practice and applications include crew scheduling (both urban and airline), vehicle routing, graph coloring and paper cutting stock. See Anbil, Tanga, and Johnson [10], Desrochers and Soumis [121], Dumas, Desrosiers, and Soumis [134], Mehrotra and Trick [327], and Vance et al. [440].

### 9.5.3 Branching on Hyperplanes

In Example 9.5 we saw that branching on a single variable and creating new candidate problems by adding constraints of the form  $x_k \leq n_k$  and  $x_k \geq n_k + 1$  led to a tree of exponential size. However, by extending branching on a single variable to branching on a hyperplane required a search tree with only two nodes. The branching strategy for type 1 special ordered sets and for set partitioning problems were two examples of branching on hyperplanes. In

this subsection the idea of hyperplane branching is applied to general integer programs. Since the decision problem, “given rational  $A, b$  does there exist a feasible integer solution to  $Ax \leq b$ ” is  $\mathcal{NP}$ -Complete, it is very unlikely that a polynomial time algorithm exists for this decision problem. However, Lenstra [285] showed that this decision problem can be solved in polynomial time *for a fixed number of variables*. This was a very surprising and interesting result. Lenstra used results from lattice theory and the geometry of numbers to develop a branch-and-bound algorithm based upon hyperplane branching.

The following lemma due to von zur Gathen and Sieveking [170] implies, without loss, that we can take the polyhedron defined by  $Ax \leq b$  to be a polytope.

**Lemma 9.7 (von zur Gathen and Sieveking)** *If  $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$  contains an integer vector it contains an integer vector of size at most  $6n^3\phi$  where  $\phi$  is the maximum of  $n$  and the size of each inequality  $a_i^T x \leq b_i$  defining  $P$ .*

As usual, by size in Lemma 9.7 we mean the binary encoding of the data. Therefore, if we want to find an integer solution in the polyhedron defined by  $Ax \leq b$  with facet complexity  $\phi$  we know  $|x_j| \leq (6n^3\phi)$  for all  $j = 1, \dots, n$  which implies  $|x_j| \leq 2^{(6n^3\phi)}$ . One possibility is to branch on variable  $x_1$  and create a set of nodes by adding the constraints

$$x_1 = -2^{(6n^3\phi)}, \quad x_1 = -2^{(6n^3\phi)} + 1, \dots, x_1 = 2^{(6n^3\phi)}.$$

Then from each of these nodes branch on variable  $x_2$ , etc. This results in a complete enumeration algorithm of order  $O((2^{(6n^3\phi)})^n)$ . Unfortunately, even for a fixed  $n$ , this is not a polynomial function of the binary encoding of the data. The fundamental problem is that

$$\max\{x_i \mid x \in P\} - \min\{x_i \mid x \in P\} \tag{9.16}$$

is not bounded by a polynomial function of the dimension  $n$  and the binary encoding of the data. However, if the objective function in (9.16) is generalized to allow any integer vector  $v$ , Khintchine [264] has shown that if the polytope contains an integer vector, then there is a direction in which the width of the polytope is bounded by a constant depending on only the dimension.

**Theorem 9.8 (Khintchine)** *If  $P \subseteq \mathbb{R}^n$  is a bounded, rational polyhedron, then either  $P \cap \mathbb{Z}^n = \emptyset$  or there exists a nonzero integer vector  $v$  such that*

$$\max\{v^T x \mid x \in P\} - \min\{v^T x \mid x \in P\} \leq \gamma_n \tag{9.17}$$

where the constant  $\gamma_n$  depends only on the dimension  $n$ .

Such an integer vector  $v$  is called a *short direction* or *short vector*. Lenstra was the first to give an algorithm with a polynomial running time in fixed  $n$  for finding the short vector, or showing that  $P \cap \mathbb{Z}^n = \emptyset$ . If  $v$  is an integer vector, it is valid to consider only the integers  $t$  such that

$$\lfloor \min\{v^\top x \mid x \in P\} \rfloor \leq t \leq \max\{v^\top x \mid x \in P\} \quad (9.18)$$

and then “branch” on the hyperplanes  $v^\top x = t$  for each possible value of  $t$ . That is, we solve the series of candidate problems  $P \cap \{x \mid v^\top x = t\}$  in reduced dimension. The process is repeated for the polytope defining each candidate problem. For  $n$  fixed, the enumeration tree is at most  $n$  deep since the dimension of the candidate problem is reduced by one at each level. Then for *fixed*  $n$  the tree has a fixed number of nodes. Therefore, the problem is to either find in polynomial time, for *fixed*  $n$ , an integer vector  $x \in P$ , or find  $v \in \gamma_n(P - P)^*$  where  $P - P$  is the body  $\{x - y \mid x, y \in P\}$  and  $K^* = \{v \mid v^\top x \leq 1, \text{ for all } x \in K\}$ . The rest of the material in this section is taken from Cook et al. [100].

Let  $P = \{x \mid Ax \leq b\}$  and define

$$F(w) := \max\{w^\top x - w^\top y \mid Ax \leq b, Ay \leq b\}. \quad (9.19)$$

$$F_1(w) := F(w)$$

$$F_i(w) := \min F(w + \alpha_1 b^1 + \alpha_2 b^2 + \cdots + \alpha_{i-1} b^{i-1}), \quad i \geq 2 \quad (9.20)$$

with the minimum over all rational  $\alpha_1, \dots, \alpha_{i-1}$ .

Recall from Subsection 4.2.1 in Chapter 4, that a lattice  $L$  in  $\mathbb{R}^n$  is any set that can be expressed as

$$G = \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q \lambda_i b^i, \lambda_i \in \mathbb{Z}, i = 1, \dots, q\} \quad (9.21)$$

where the  $b^1, \dots, b^q$  are rational vectors which are the basis for the lattice. When  $b^i = e^i$ , the  $i$ th unit vector for  $i = 1, \dots, n$ , the lattice generated is the set of all integer vectors in  $\mathbb{R}^n$ . If  $b^1, b^2, \dots, b^n$  is a basis for the lattice of integer vectors in  $\mathbb{R}^n$ , then any integer vector can be written as  $w = \lambda_1 b^1 + \cdots + \lambda_n b^n$  for integer  $\lambda_i$ . If  $F(w)$  is the “norm” that we select, then the nonzero vector  $v$  in the lattice which minimizes  $F(w)$  is the short vector we are seeking in (9.17). The rest of this subsection is devoted to finding an approximation of the shortest vector in the lattice.

**Proposition 9.9** For  $i = 2, \dots, n$ ,

$$\begin{aligned} F_i(w) &= \max\{w^\top x - w^\top y \mid Ax \leq b, Ay \leq b, \\ &\quad (b^1)^\top(x - y) = 0, \dots, (b^{i-1})^\top(x - y) = 0\}. \end{aligned} \quad (9.22)$$

**Proof:**

$$\begin{aligned} F_i(w) &= \min_{\alpha} F(w + \alpha_1 b^1 + \alpha_2 b^2 + \dots + \alpha_{i-1} b^{i-1}) \\ &= \min_{\alpha} \max\{(w + \alpha_1 b^1 + \dots + \alpha_{i-1} b^{i-1})^\top(x - y) \mid Ax \leq b, Ay \leq b\} \\ &= \min_{\alpha, t, u} b^\top(t + u) \\ &\text{s.t.} \\ &\quad A^\top t - \alpha_1 b^1 - \dots - \alpha_{i-1} b^{i-1} = w \\ &\quad A^\top u + \alpha_1 b^1 + \dots + \alpha_{i-1} b^{i-1} = -w \\ &\quad t, u \geq 0 \\ &= \max w^\top(x - y) \\ &\text{s.t.} \\ &\quad Ax \leq b, Ay \leq b \\ &\quad (b^1)^\top(x - y) = 0, \dots, (b^{i-1})^\top(x - y) = 0 \quad \square \end{aligned}$$

Thus, each  $F_i(w)$  can be computed by solving a single linear program! The significance of the definition of the  $F_i(w)$  is given in the following theorem.

**Proposition 9.10** If  $b^1, \dots, b^n$  is a basis for the lattice of integer vectors in  $\mathbb{R}^n$  such that  $F_1(b^1) \leq F_2(b^2) \leq \dots \leq F_n(b^n)$ , then  $b^1$  achieves the minimum value of  $F(\bullet)$  over all nonzero integer vectors in  $\mathbb{R}^n$ .

**Proof:** Let  $w$  be an arbitrary nonzero integer vector  $\mathbb{R}^n$ . Then  $w = \lambda_1 b^1 + \dots + \lambda_n b^n$  where the  $\lambda_i$  are integer. Let  $k$  be the largest index such that  $\lambda_k \neq 0$ . Then

$$F(w) = F_1(w) \geq F_k(w) = F_k(\lambda_k b^k) = |\lambda_k| F_k(b^k) \geq F_1(b^1) = F(b^1).$$

The first inequality follows from the fact that  $F_k(w)$  is defined from  $F_1(w)$  by adding constraints. The first equality follows from

$$\begin{aligned} F_k(w) &= F_k(\lambda_1 b^1 + \dots + \lambda_n b^n) \\ &= F_k(\lambda_1 b^1 + \dots + \lambda_k b^k) \\ &= F_k(\lambda_k b^k) \text{ since } (b^j)^\top(x - y) = 0, j = 1, \dots, k - 1. \end{aligned}$$

Since  $\lambda_k$  is an integer  $|\lambda_k|F_k(b^k) \geq F_1(b^1)$ .  $\square$

**Definition** Let  $0 < \epsilon < 1/2$ , then the basis  $b^1, \dots, b^n$  is reduced if and only if

1.  $F_i(b^{i+1} + \mu b^i) \geq F_i(b^{i+1})$  for all integers  $\mu$ ; (do not confuse this  $\mu$  with the barrier parameter in Chapter 8)
2.  $F_i(b^{i+1}) \geq (1 - \epsilon)F_i(b^i)$ .

**Proposition 9.11** If  $b^1, \dots, b^n$  is a reduced basis for  $\mathcal{Z}^n$ , then  $F_{i+1}(b^{i+1}) \geq ((1/2) - \epsilon)F_i(b^i)$  for  $i = 1, \dots, n - 1$ .

**Proof:** By definition of  $F_i(\bullet)$ ,

$$\begin{aligned} F_{i+1}(b^{i+1}) &= \min_{\alpha_1, \dots, \alpha_i} F(b^{i+1} + \alpha_1 b^1 + \dots + \alpha_i b^i) \\ &= \min_{\alpha_i} \min_{\alpha_1, \dots, \alpha_{i-1}} F(b^{i+1} + \alpha_i b^i + (\alpha_1 b^1 + \dots + \alpha_{i-1} b^{i-1})) \\ &= \min_{\alpha} F_i(b^{i+1} + \alpha b^i). \end{aligned}$$

If  $\bar{\alpha}$  is the optimal  $\alpha$ , then  $\bar{\mu} = \bar{\alpha} + f$  where  $f \in (0, 1/2]$  and  $\bar{\mu}$  is an integer. By Proposition 9.9,  $F_i(b^{i+1} + \theta b^i)$  is a maximization linear program which is convex in the parameter  $\theta$ . Then

$$\begin{aligned} F_{i+1}(b^{i+1}) &= F_i(b^{i+1} + \bar{\alpha} b^i) \\ &= F_i(b^{i+1} + \bar{\mu} b^i - f b^i) \quad \text{and} \\ F_i(b^{i+1} + \bar{\mu} b^i - f b^i) &\geq F_i(b^{i+1} + \bar{\mu} b^i) - F_i(f b^i) \quad \text{which implies} \\ F_i(b^{i+1} + \bar{\alpha} b^i) &\geq F_i(b^{i+1} + \bar{\mu} b^i) - f F_i(b^i) \\ &\geq \min \{F_i(b^{i+1} + \mu b^i) \mid \mu \in \mathcal{Z}\} - f F_i(b^i) \end{aligned}$$

Then,

$$\begin{aligned} F_i(b^{i+1} + \bar{\alpha} b^i) &\geq \min \{F_i(b^{i+1} + \mu b^i) \mid \mu \in \mathcal{Z}\} - f F_i(b^i) \\ &\geq \min \{F_i(b^{i+1} + \mu b^i) \mid \mu \in \mathcal{Z}\} - (1/2)F_i(b^i) \end{aligned}$$

since  $F_i(b^i)$  is nonnegative. Then from part 1 of the definition of reduced basis

$$F_{i+1}(b^{i+1}) \geq F_i(b^{i+1}) - (1/2)F_i(b^i).$$

and from part 2 of the definition of reduced basis

$$F_{i+1}(b^{i+1}) \geq (1 - \epsilon)F_i(b^i) - (1/2)F_i(b^i) = ((1/2) - \epsilon)F_i(b^i) \quad \square$$

**Corollary 9.12** Let  $b^1, \dots, b^n$  be a reduced basis for  $\mathbb{Z}^n$ , then

$$F(w) \geq ((1/2) - \epsilon)^{n-1} F(b^1)$$

for all nonzero integer vectors  $w$  in  $\mathbb{R}^n$ .

**Proof:** Mimic the proof of Proposition 9.10 and show that for any nonzero integer vector  $w$  there is an integer  $k \leq n$  such that  $F(w) \geq F_k(b^k)$ . Since  $k \leq n$ , repeated application of Proposition 9.11 yields  $F(w) \geq ((1/2) - \epsilon)^{n-1} F(b^1)$ .  $\square$

By Corollary 9.12, if we can find a reduced basis for the lattice of integer vectors in  $\mathbb{Z}^n$ , then we at least have an  $\epsilon$ -approximation of the shortest vector in this lattice relative to  $F(\bullet)$  norm. The definition of reduced basis suggests how one can find a reduced basis starting with the basis  $e^i$ ,  $i = 1, \dots, n$  of unit vectors. The algorithm is based on the following corollary which states that performing elementary column operations (see Chapter 4) on a basis does not change the lattice it generates.

**Corollary 9.13** Let  $A$  and  $A'$  be nonsingular matrices. Then the following are equivalent:

1. the columns of  $A$  and  $A'$  generate the same lattice;
2.  $A'$  comes from  $A$  by elementary column operations.

#### Algorithm 9.14 (Reduced Basis Algorithm)

**Step 1: (Initialization)**  $i \leftarrow 1$  and  $b^i \leftarrow e^i$ ,  $i = 1, \dots, n$

**Step 2:** Replace  $b^{i+1}$  by  $b^{i+1} + \mu b^i$ , where  $\mu$  is the integer that minimizes  $F_i(b^{i+1} + \mu b^i)$ . (In practice we observe  $F_{i+1}(b^{i+1}) = \min_\alpha F_i(b^{i+1} + \alpha b^i)$  and solve the linear program defined by  $F_{i+1}(b^{i+1})$  given in (9.22). The dual variable associated with the constraint  $b^i x - b^i y = 0$  gives the optimal  $\bar{\alpha}$ . Since  $F_i(b^{i+1} + \alpha b^i)$  is convex in  $\alpha$  the optimal  $\mu$  is either  $\lfloor \bar{\alpha} \rfloor$  or  $\lceil \bar{\alpha} \rceil$ .)

**Step 3:** If  $F_i(b^{i+1}) < (1 - \epsilon)F_i(b^i)$  swap  $b^i$  and  $b^{i+1}$  and set  $i$  to the maximum of 1 and  $i - 1$ , else set  $i \leftarrow i + 1$ . If  $i \leq n$ , go to Step 2; otherwise, stop. (In this step it is not necessary to evaluate  $F_i(b^{i+1})$ . In the prior step  $b^{i+1}$  was replaced by either  $b^{i+1} + \lfloor \bar{\alpha} \rfloor b^i$  or  $b^{i+1} + \lceil \bar{\alpha} \rceil b^i$ ) and we evaluated both  $F_i(b^{i+1} + \lfloor \bar{\alpha} \rfloor b^i)$  and  $F_i(b^{i+1} + \lceil \bar{\alpha} \rceil b^i)$ .)

Upon termination of this algorithm  $b^1$  is an approximation of the shortest nonzero vector in the lattice and is used as the branching hyperplane. Lovász and Scarf [294] show that the reduced basis algorithm runs in polynomial time for fixed  $n$ . It is left as an exercise to show that this implies the existence of a fixed  $n$  polynomial algorithm for the decision problem “given rational  $A, b$  does there exist a feasible integer solution to  $Ax \leq b$ ?”

**Example 9.15** Consider the following integer program.

$$\begin{array}{rcl} x_1 + x_2 & \geq & 9 \\ 2x_1 + 2x_2 & \leq & 23 \\ -x_1 + x_2 & \leq & 9 \\ x_1 - x_2 & \leq & 9 \\ x_1, x_2 & \geq & 0 \end{array}$$

Begin iteration 1 with the initial basis  $[1 \ 0]^T$  and  $[0 \ 1]^T$  and  $\epsilon = 0.3$ . Solve the linear program  $F_2(b^2)$ .

```

MAX      X2 - Y2
SUBJECT TO
 2)  X1 + X2 >= 9
 3)  2 X1 + 2 X2 <= 23
 4)  - X1 + X2 <= 9
 5)  X1 - X2 <= 9
 6)  Y1 + Y2 >= 9
 7)  2 Y1 + 2 Y2 <= 23
 8)  - Y1 + Y2 <= 9
 9)  Y1 - Y2 <= 9
10)  X1 - Y1 = 0
END
LP OPTIMUM FOUND AT STEP      3
OBJECTIVE VALUE =   2.50000000

```

The optimal dual variable on the constraint  $x_1 - y_1 = 0$  is 1.0. This implies that the  $\mu$  which minimizes  $F_1(b^2 + \mu b^1)$  is 1.0. Replace  $b^2$  with  $b^2 + \mu b^1$ . This gives  $b^2 = [1 \ 1]^T$ . Calculate  $F_1(b^1)$ .

```

MAX      X1 - Y1
SUBJECT TO

```

```

2) X1 + X2 >= 9
3) 2 X1 + 2 X2 <= 23
4) - X1 + X2 <= 9
5) X1 - X2 <= 9
6) Y1 + Y2 >= 9
7) 2 Y1 + 2 Y2 <= 23
8) - Y1 + Y2 <= 9
9) Y1 - Y2 <= 9
END
LP OPTIMUM FOUND AT STEP      2
OBJECTIVE VALUE =   10.2500000

```

Since  $b^2 = b^2 + \mu b^1$ ,  $F_1(b^2) = F_1(b^2 + \mu b^1) = 2.5 < (1 - \epsilon) * F_1(b^1) = (0.7)10.25$  swap  $b^1$  and  $b^2$  and keep  $i = 1$ . The new basis is  $[1 \ 1]^\top$  and  $[1 \ 0]^\top$ . We are now at iteration 2 with  $i = 1$ . Find  $F_2(b^2)$ .

```

MAX      X1 - Y1
SUBJECT TO
2) X1 + X2 >= 9
3) 2 X1 + 2 X2 <= 23
4) - X1 + X2 <= 9
5) X1 - X2 <= 9
6) Y1 + Y2 >= 9
7) 2 Y1 + 2 Y2 <= 23
8) - Y1 + Y2 <= 9
9) Y1 - Y2 <= 9
10) X1 + X2 - Y1 - Y2 = 0
END
LP OPTIMUM FOUND AT STEP      3
OBJECTIVE VALUE =   9.00000000

```

The value of the optimal dual variable on the constraint  $x_1 + x_2 - y_1 - y_2$  is -0.5. Round -0.5 down to -1.0 and evaluate  $F_1(b^2 - b^1)$ .

```

MAX      - X2 + Y2
SUBJECT TO
2) X1 + X2 >= 9
3) 2 X1 + 2 X2 <= 23
4) - X1 + X2 <= 9
5) X1 - X2 <= 9

```

```

6) Y1 + Y2 >= 9
7) 2 Y1 + 2 Y2 <= 23
8) - Y1 + Y2 <= 9
9) Y1 - Y2 <= 9
END
LP OPTIMUM FOUND AT STEP      2
OBJECTIVE VALUE =   10.2500000

```

Round -0.5 up to 0 and evaluate  $F_1(b^2 + 0b^1)$ .

```

MAX      X1 - Y1
SUBJECT TO
2) X1 + X2 >= 9
3) 2 X1 + 2 X2 <= 23
4) - X1 + X2 <= 9
5) X1 - X2 <= 9
6) Y1 + Y2 >= 9
7) 2 Y1 + 2 Y2 <= 23
8) - Y1 + Y2 <= 9
9) Y1 - Y2 <= 9
END
LP OPTIMUM FOUND AT STEP      1
OBJECTIVE VALUE =   10.2500000

```

There is tie between  $\mu = -1.0$  and  $\mu = 0.0$ . Take  $\mu = -1.0$  and  $b^2 = b^2 - b^1 = [0 \ -1]^\top$ . Evaluate  $F_1(b^1)$ .

```

MAX      X1 + X2 - Y1 - Y2
SUBJECT TO
2) X1 + X2 >= 9
3) 2 X1 + 2 X2 <= 23
4) - X1 + X2 <= 9
5) X1 - X2 <= 9
6) Y1 + Y2 >= 9
7) 2 Y1 + 2 Y2 <= 23
8) - Y1 + Y2 <= 9
9) Y1 - Y2 <= 9
END
LP OPTIMUM FOUND AT STEP      1
OBJECTIVE VALUE =   2.5000000

```

Since  $F_1(b^2) = 10.25$ ,  $F_1(b^2) \not< (1 - \epsilon)F_1(b^1)$ . Set  $i = 2$  and terminate with the reduced basis of  $[1 \ 1]$  and  $[0 \ -1]$ . The approximate shortest vector is  $[1 \ 1]^\top$ . How many branches are required at the first level of the enumeration tree?

## 9.6 CONCLUSION

The use of integer variables permits many new modeling possibilities. Indeed, integer variables are required in order to realistically model many problems in engineering and business. However, the use of integer variables also makes it much more difficult to optimize the model. Linear programming based branch-and-bound was the first truly practical method for solving integer linear programs. Despite using many of the variable selection and branching strategies described in this chapter, linear programming based branch-and-bound will fail for many realistic sized problems. In Chapter 12 we study methods for improving on the bounds generated by linear programming relaxations. In Chapters 15 and 16 we study methods for improving the model formulation before using linear programming based branch-and-bound.

## 9.7 EXERCISES

- 9.1 Repeat Example 9.4 using  $x_2$  as the branching variable for the first branch.
- 9.2 In answering this question assume that variable  $x_j$  is a general integer variable and has fractional value  $\bar{x}_j = n_j + f_j$  in the linear programming relaxation of a candidate problem. Create two candidate problems from the constraints  $x_j \geq n+1$  and  $x_j \leq n$ . Is it possible that in some future descendant of this candidate problems variable  $x_j$  becomes fractional again?
- 9.3 Model with 0/1 variables the condition that “project  $j$  is funded if and only if projects  $k$  and  $l$  are funded.”
- 9.4 Model with 0/1 variables the condition that “if projects  $k$  and  $l$  are funded, then project  $j$  cannot be funded.”
- 9.5 Construct two polyhedrons  $\Gamma_1 = \{x \mid A_1x \leq b^1\}$  and  $\Gamma_2 = \{x \mid A_2x \leq b^2\}$  such that there does not exist a value of  $M$  such that the modeling (9.6)-(9.8) represents the disjunction  $\Gamma_1 \cup \Gamma_2$ .

9.6 This exercise is taken from Shapiro [405]. A coal supplier has contracted to sell coal at a fixed price to customers in three separate locations over the next three time periods. The coal is to be supplied from mines at two other locations, each mine with its own extraction costs which are a function of the cumulative extraction at the mine over time. See Table 9.1. The supplier wishes to meet his/her contracts at minimal total extraction and transportation costs. See Tables 9.2 and 9.3. Due to restrictions on the availability of labor and equipment at each mine, only 10 tons may be extracted from mine one during any time period, and eight tons from mine two during any time period. Discount future costs at a rate of 25%.

**Table 9.1** Coal Extraction Costs

		Mine at Location One		
Cumulative Extraction	Marginal Costs	0-8 tons	8-16 tons	16-32 tons
		\$1 ton	\$4 ton	\$6 ton
			Mine at Location Two	
Cumulative Extraction	Marginal Costs	0-4 tons	4-20 tons	20-40 tons
		\$2 ton	\$3 ton	\$4 ton

**Table 9.2** Coal Demand in Tons

Demand Location	Period 1	Period 2	Period 3
1	2	3	5
2	4	4	4
3	8	5	1

**Table 9.3** Transportation Costs \$ per Ton

	Customer	Customer	Customer
Mine 1	2	3	7
Mine 2	5	4	2

- Formulate a linear program for finding the minimal total extraction and transportation costs while satisfying the demand requirements.
- Formulate a linear integer program for finding the minimal total extraction and transportation costs while satisfying the demand requirements using the modified extraction costs in Table 9.4.

**Table 9.4** Modified Coal Extraction Costs

Cumulative Extraction Marginal Costs	Mine at Location One		
	0-8 tons	8-16 tons	16-32 tons
Cumulative Extraction Marginal Costs	\$10 ton	\$8 ton	\$6 ton
	Mine at Location Two		
Cumulative Extraction Marginal Costs	0-4 tons	4-20 tons	20-40 tons
	\$12 ton	\$6 ton	\$4 ton

9.7 The Haleiwa Surf Shop sells two types of surfboards. The first board is the Sunset Special. The Sunset Special is a heavy, thick and wide board (a tanker), 88 inches long. It is designed for beginners, but is also appropriate for intermediates looking for a slow board on bigger waves. The Surf Shop buys these boards from their distributor at \$50 dollars a board.

The second board is the Waimea Deluxe. It is a smaller board, only 74 inches long, and is designed for big wave riders. It is also appropriate for intermediates wanting to learn with a faster board on small waves. The Surf Shop shapes these boards themselves at a marginal cost of \$90.

The Surf Shop estimates that in Haleiwa there are about 30 beginning surfers looking for boards, 100 intermediate level surfers and 20 advanced big wave riders. In Table 9.5 is a list of reservation prices for each customer type. The reservation price is the maximum price a customer is willing to pay for a product. For example, a beginner is willing to pay up to \$150 for the Sunset Special. An intermediate is willing to pay at most \$400 for the bundle, i.e. both the Sunset Special and the Waimea Deluxe.

The Surf Shop wants to set a price for each board and the bundle (i.e. a price for both boards) which will maximize their profit. In setting these prices they must take into account the fact that each customer will buy the board (or the bundle) that maximizes his or her consumer surplus. Consumer surplus is defined as the reservation price minus the price of the product. For example, assume the Surf Shop decides to price the Sunset Special at \$150, the Waimea Deluxe at \$240 and the bundle at \$375. Then the intermediate level surfer will purchase the Waimea Deluxe since this gives a consumer surplus of \$60. (The consumer surplus on the Sunset Special is  $-\$50 = \$100 - \$150$ , and the surplus on the bundle is  $\$25 = \$400 - \$375$ .) An item with negative surplus is never purchased.

Formulate a mixed integer linear programming model to solve this profit maximization problem. Hint: you should have continuous variables which define the price of each item and 0/1 variables for each customer - item combination.

**Table 9.5 Reservation Prices**

Customer	Sunset Special	Waimea Deluxe	Bundle
Beginner	150	200	300
Intermediate	100	300	400
Advanced	0	400	400

9.8 Use the basis reduction algorithm to find an approximate short vector for the following linear program.

$$\begin{aligned} \min \quad & x_0 \\ \text{s.t.} \quad & x_0 + 2 \sum_{j=1}^6 x_j = 3 \\ & x_0, x_j \in \{0, 1\}, j = 1, \dots, 6 \end{aligned}$$

9.9 Consider the pure 0/1 linear program

$$\min \{c^\top x | Ax \geq b, x_j \in \{0, 1\} \text{ } j = 1, \dots, n\}.$$

Carefully explain how you would modify a linear programming based branch-and-bound in order to obtain the  $K$  best solutions.

9.10 Prove that the existence of an algorithm (or subroutine) which can solve linear integer programs in polynomial time for fixed  $n$  (where  $n$  is the number of variables) implies the existence of an algorithm for integer programming which is polynomial for fixed  $m$  where  $m$  is the number of rows.

9.11 Prove Corollary 9.13.

9.12 Assume that a reduced basis can be found in polynomial time for fixed  $n$ . Show that this implies the existence of a polynomial algorithm for fixed  $n$  for deciding if there is an integer vector in the polyhedron  $\{x | Ax \geq b\}$ .

9.13 Does the existence of a polynomial time reduced basis algorithm imply the existence of a polynomial time algorithm for deciding if there is an integer vector in the polyhedron  $\{x | Ax \geq b\}$ ?

9.14 Prove Proposition 9.6.

9.15 The “Oldie-But-Goodie” company is in the process of releasing a cassette tape with 10 songs on it. The length of tape required for each song (in centimeters) is given in the table below.

Song	Length
"In the Still of the Night"	54
"The Closer You Are"	68
"The Duke of Earl"	73
"Deep in My Heart"	45
"Sincerely"	29
"One Summer Night"	56
"Ship of Love"	35
"My True Story"	37
"Tonight, Tonight"	55
"Earth Angel"	71
	Total = 523

The tape will have two sides (side A and side B). The objective is to assign songs to each side so that the total length of the tape is minimized. The length of the tape will equal the length of side A which will equal the length of side B. Assume side A will have no slack tape, but side B may. Formulate an integer linear program for making the song assignment in order to minimize the tape length. Students who can name the group that did each song receive extra credit.

- 9.16 Bill E. Solestes owns four commodities which he must store in seven different silos. Each silo can contain *at most* one commodity. Associated with each commodity and silo combination is a loading cost. Each silo has a finite capacity so some commodities may have to be split over several silos. The table contains the data for this problem.

Commodity	Loading Cost per Ton						
	Silo						
	1	2	3	4	5	6	7
A	1	2	2	3	4	5	5
B	2	3	3	3	1	5	5
C	4	4	3	2	1	5	5
D	1	1	2	2	3	5	5

The capacity (in tons) for the seven silos is, respectively, 25, 25, 40, 60, 80, 100, and 100. There are 75 tons of commodity A, 50 tons of commodity B, 25 tons of commodity C and 80 tons of commodity D which must be stored in the silos. Formulate a mixed integer linear program for finding the minimum cost storage plan. This problem is called the *segregated storage problem* in the literature.

- 9.17 Assume you are given five items A, B, C, D, and E to schedule on a single machine. The time it takes to produce each item is composed of two parts. The first time component is the time it takes to adjust the machine in order to make it compatible with the item being produced. These times (in hours) are sequence dependent and given in the table below.<sup>3</sup>

	A	B	C	D	E
A	0	2	5	3	7
B	3	0	2	4	4
C	2	1	0	4	5
D	1	3	3	0	5
E	5	5	2	2	0

For example if job B is processed immediately after job A it takes 2 hours to adjust the machine. If job E is processed immediately after job D it takes 5 hours to adjust the machine. Note that the times are not symmetric so if D is processed immediately after job E it takes only 2 hours.

After the machine is adjusted properly for the respective job it takes 3 hours to process job A, 2.5 hours to process job B, 9 hours to process job C, 3.7 hours to process job D and 5 hours to process job E. Formulate a mixed integer linear programming model to find the sequence of jobs which will minimize the total production time (machine adjustment plus job processing times).

- 9.18 Show that if  $f(x)$  is piecewise linear and convex then equations (9.4) and (9.5) are not needed when using variable definition (9.1)-(9.2).

---

<sup>3</sup>This problem is taken from Schrage [402]

## **PART IV**

---

# **SOLVING LARGE SCALE PROBLEMS: DECOMPOSITION METHODS**

---

## PROJECTION: BENDERS' DECOMPOSITION

### 10.1 INTRODUCTION

A generic structure found in many distribution, routing, and location applications is

$$\min c^T x + f^T y \quad (10.1)$$

$$\text{s.t. } Ax + By \geq b \quad (10.2)$$

$$y \in Y \quad (10.3)$$

$$x \geq 0 \quad (10.4)$$

where the decision variables have been partitioned into two sets of variables  $x \in \mathbb{R}^{n_1}$  and  $y \in \mathbb{R}^{n_2}$ . In particular, assume that the  $A$  matrix has very special structure so the problem in the  $x$  variables only, is a relatively “easy” problem. For example, if the  $y$  variables are fixed at  $y = \bar{y}$ ,  $Ax \geq b - B\bar{y}$  might be the constraint set for a transportation problem. See model (TLP) in Subsection 1.3.5 of Chapter 1. The constraint set  $y \in Y$ , might be a polyhedron, but it could also be a set with discrete variables, or a set with nonlinearities.

One way to attack this problem is to project out the  $x$  variables and then solve the  $y$  variable problem that remains. There are two difficulties with this approach. The first problem was described in Chapter 2. Projecting out the  $x$  variables will generally result in a system where the number of constraints is an exponential function of the number of  $x$  variables. The second problem is that projection takes no advantage of the special structure that might be present in the  $A$  matrix. In this chapter we describe a projection algorithm, due to J.F. Benders [49], which overcomes both of these problems. The  $x$  variables are projected out and replaced with more constraints but these constraints are

added “on the fly,” that is, on an as needed basis. Furthermore, the Benders’ method for generating these constraints takes advantage of the special structure of the  $A$  matrix. The Benders’ projection algorithm is often called Benders’ decomposition. The word decomposition stemming from the fact that the variables have been “decomposed” into a set of  $x$  variables and a set of  $y$  variables. Later, in Chapter 16, Benders’ decomposition is used once again. It is used in Chapter 16 to solve integer programming models that have been reformulated using auxiliary variables. The auxiliary variables are projected out using a Benders’ like algorithm to get valid cutting planes in the space of the original variables.

The basic Benders’ algorithm is outlined in Section 10.2. This algorithm is illustrated in detail with the simple plant location problem in Section 10.3. A key aspect of the Benders’ algorithm is finding a dual solution to a linear programming subproblem. This is the topic of Section 10.4. Concluding remarks are given in Section 10.5. Exercises are provided in Section 10.6.

## 10.2 THE BENDERS’ ALGORITHM

This development is based on Propositions 2.22 and 2.23 from Chapter 2. These propositions are restated below for completeness.

**Proposition 10.1** *If  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ , then*

$$\text{proj}_y(P) = \{y \in \mathbb{R}^{n_2} \mid (u^i)^\top By \geq u^i b, i = 1, \dots, r\}$$

where  $u^1, \dots, u^r$  are the extreme rays of the projection cone

$$C_x(P) = \{u \mid A^\top u = 0, u \geq 0\}.$$

**Proposition 10.2** *If  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq b\}$ , and  $u^i, i = 1, \dots, q$  are the multipliers that are generated using projection, then the extreme rays of the projection cone  $C_x(P) = \{u \mid A^\top u = 0, u \geq 0\}$  are contained in this set of multipliers.*

The original problem (10.1)-(10.4) is equivalent to

$$\min z_0$$

$$\begin{aligned} \text{s.t. } z_0 - c^T x &\geq f^T y \\ Ax &\geq b - By \\ x &\geq 0 \\ y &\in Y \end{aligned}$$

Assign the multiplier  $u_0$  to the constraint  $z_0 - c^T x \geq f^T y$ , the vector of multipliers  $u$  to the constraints  $Ax \geq b - By$  and the vector of multipliers  $w$  to the constraints  $x \geq 0$ . Using these multipliers, project out the  $x$  variables. This gives

$$\begin{aligned} \min \quad & z_0 \\ \text{s.t. } & u_0^i z_0 \geq u_0^i f^T y + (u^i)^T (b - By), \quad i = 1, \dots, r \\ & y \in Y. \end{aligned}$$

Since the multipliers  $(u_0^i, u^i, w^i)$  can be rescaled without loss, assume  $u_0^i = 1, i = 1, \dots, q$  and  $u_0^i = 0, i = q+1, \dots, r$ . Then the projected system is

$$\min z_0 \tag{10.5}$$

$$(BMP) \quad \text{s.t. } z_0 \geq f^T y + (u^i)^T (b - By), \quad i = 1, \dots, q \tag{10.6}$$

$$0 \geq (u^i)^T (b - By), \quad i = q+1, \dots, r \tag{10.7}$$

$$y \in Y. \tag{10.8}$$

Problem (BMP) is the Benders' *master program*. From the theory of projection  $\bar{y}$  is an optimal solution to (BMP) if and only if there is a  $\bar{x}$  such that  $(\bar{x}, \bar{y})$  is an optimal solution to the original problem. By Proposition 10.2 the extreme rays of the projection cone  $C_x(P) = \{(u_0, u, w) \mid A^T u + w - cu_0 = 0, (u_0, u, w) \geq 0\}$  are contained in the set of vectors  $\{(u_0^i, u^i, w^i) \mid i = 1, \dots, r\}$  generated by projection. By Proposition 10.1, only the extreme rays of the projection cone  $C_x(P)$  are needed to generate the inequalities in (10.6)-(10.7) which characterize the projection into  $y$  space. Therefore, we can assume that the inequalities in (10.6)-(10.7) are generated from the extreme rays of  $C_x(P)$ . However, even considering only extreme rays, the Benders' master program clearly has far too many constraints to optimize directly. The key insight of Benders is to solve a *relaxed master* by keeping only a very small subset of constraints in (10.6)-(10.7). Then, test if there is some constraint not in the relaxed master that is violated by the solution to the relaxed master. If so, update the relaxed master by adding the violated constraint. This process is often called constraint generation or adding constraints on the fly.

A systematic method is needed for finding constraints in the Benders' master that violate the solution of the relaxed master. Since the master program

contains too many constraints to test each constraint individually, a better method is required for finding a violated constraint.

**Proposition 10.3** *If  $\{(u_0^i, u^i, w^i) \mid i = 1, \dots, r\}$  are the extreme rays of the projection cone  $C_x = \{(u_0, u, w) \mid A^\top u + w - cu_0 = 0, (u_0, u, w) \geq 0\}$  scaled so that  $u_0^i = 1$ ,  $i = 1, \dots, q$  and  $u_0^i = 0$ ,  $i = q+1, \dots, r$ , then  $u^i$ ,  $i = 1, \dots, q$  are the extreme points of the polyhedron  $\{u \mid A^\top u \leq c, u \geq 0\}$  and  $u^i$ ,  $i = q+1, \dots, r$  are the extreme rays of the associated recession cone  $\{u \mid A^\top u \leq 0, u \geq 0\}$ .*

**Proof:** Observe that  $u^i$  is an extreme point of the polyhedron  $\{u \mid A^\top u \leq c, u \geq 0\}$  if and only if  $(1, u^i)$  is an extreme ray of the cone  $\{(u_0, u) \mid A^\top u - cu_0 \leq 0, (u_0, u) \geq 0\}$ .  $\square$

The significance of Proposition (10.3) is that if  $(\bar{z}_0, \bar{y})$  is a feasible solution to a relaxed Benders' master we can find a constraint that violates the master program by solving the linear program  $\max\{f^\top \bar{y} + (b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\}$ . This linear program is the *Benders' subproblem*. If  $u^i$  is an optimal solution to the Benders' subproblem and  $\bar{z}_0 < f^\top \bar{y} + (\bar{u})^\top (b - B\bar{y})$  then add the constraint  $z_0 \geq f^\top y + (u^i)^\top (b - By)$  to the relaxed master. If the linear program  $\max\{f^\top \bar{y} + (b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\}$  is unbounded then there is an extreme ray  $u^i$  in the cone  $\{u \mid A^\top u \leq 0, u \geq 0\}$  such that  $(u^i)^\top (b - B\bar{y}) > 0$ . In this case add the constraint  $(u^i)^\top (b - By) \leq 0$  to the Benders' relaxed master. Iterate in this fashion. What does it mean if the linear program  $\max\{f^\top \bar{y} + (b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\}$  is infeasible?

By solving the Benders' relaxed master and then the Benders' subproblem it is possible to generate a sequence of lower and upper bounds on the optimal solution value to the original problem. Because the Benders' relaxed master is a relaxation of the full Benders' master, and we are assuming a minimization problem, any feasible solution  $(\bar{z}_0, \bar{y})$  to the relaxed master provides a *lower bound*, LB, on the optimal solution value to (10.1)-(10.4). The lower bound is  $LB = \bar{z}_0$ . Adding constraints to the relaxed master can only help this lower bound, i.e. make it larger. Solving the Benders' subproblem provides an upper bound, UB, on the optimal solution value to (10.1)-(10.4). This is stated formally in Lemma 10.4.

**Lemma 10.4** *If  $\bar{y} \in Y$  and there is an optimal solution to the Benders' subproblem  $\max\{f^\top \bar{y} + (b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\}$ , then this optimal solution value is an upper bound on the optimal solution value of (BMP).*

**Proof:** Since  $f^\top \bar{y}$  is a constant term in the Benders' subproblem objective function, if there is an optimal solution  $u^i$  to the Benders' subproblem by strong duality

$$\begin{aligned} & \max\{f^\top \bar{y} + (b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\} \\ &= f^\top \bar{y} + \max\{(b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\} \\ &= f^\top \bar{y} + \min\{c^\top x \mid Ax \geq b - B\bar{y}, x \geq 0\}. \end{aligned}$$

If  $\bar{x}$  is an optimal solution to the linear program  $\min\{c^\top x \mid Ax \geq b - B\bar{y}, x \geq 0\}$ , then  $(\bar{x}, \bar{y})$  is a feasible solution to the original problem (10.1)-(10.4). Therefore  $f^\top \bar{y} + (b - B\bar{y})^\top u^i = f^\top \bar{y} + c^\top \bar{x}$  is the value of a feasible solution for (10.1)-(10.4) which is a valid upper bound on the optimal value of the equivalent problem (BMP).  $\square$

In the following sections we also refer to the linear program  $f^\top \bar{y} + \min\{c^\top x \mid Ax \geq b - B\bar{y}, x \geq 0\}$  as the Benders' subproblem. By generating a sequence of upper and lower bounds one can terminate when  $UB - LB$  is less than some  $\epsilon$  tolerance.

#### Algorithm 10.5 (Benders' Decomposition)

**Step 1:** Find an initial  $\bar{y} \in Y$  and set the upper  $UB \leftarrow \infty$  and the lower bound  $LB \leftarrow -\infty$ .

**Step 2:** For the current  $\bar{y}$  solve the Benders' subproblem  $\max\{(b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\}$ . If the subproblem is infeasible terminate (see Exercise 10.2). If the subproblem is unbounded generate an extreme direction  $u^i$ . If the subproblem has an optimal solution, find an optimal extreme point solution  $u^i$ . When an optimal solution is found set  $UB \leftarrow \min\{UB, f^\top \bar{y} + (b - B\bar{y})^\top u^i\}$  since we have just found a new feasible solution. If  $UB - LB < \epsilon$  terminate, if not go to Step 3.

**Step 3:** If  $u^i$  is an extreme point add the constraint (sometimes called a Benders' cut)  $z_0 \geq f^\top y + (u^i)^\top (b - By)$  to the relaxed master. If  $u^i$  is an extreme ray for the unbounded subproblem add  $(u^i)^\top (b - Bx) \leq 0$  to the relaxed master. Reoptimize the relaxed master and obtain a new solution  $\bar{y}$  and objective function value  $\bar{z}_0$ . Set  $LB \leftarrow \bar{z}_0$ . If  $UB - LB \leq \epsilon$ , stop, otherwise return to Step 2 with the new  $\bar{y}$ . If at any point the relaxed master becomes infeasible terminate, problem (P) is infeasible.

### 10.3 A LOCATION APPLICATION

The following is a formulation for the simple (or uncapacitated) plant location problem. This model was first introduced in Subsection 1.3.5 in Chapter 1 as the lock box location problem.

$$(SPL) \quad \begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} + \sum_{i=1}^n f_i y_i \\ \text{s.t.} \quad & \sum_{i=1}^n x_{ij} \geq 1, \quad j = 1, \dots, m \\ & -x_{ij} + y_i \geq 0, \quad i = 1, \dots, n, j = 1, \dots, m \\ & x_{ij} \geq 0, \quad i = 1, \dots, n, j = 1, \dots, m \\ & y_i \in \{0, 1\}, \quad i = 1, \dots, n. \end{aligned}$$

In this formulation  $x_{ij}$  is the fraction of customer  $j$  demand supplied by plant  $i$  and  $y_i = 1$  if plant  $i$  is open and zero otherwise. In the objective function  $f_i$  is the annual fixed cost of operating or locating a plant at location  $i$  and  $c_{ij}$  is the cost of satisfying the annual demand of customer  $j$  from location  $i$ . The constraints  $\sum_{i=1}^n x_{ij} \geq 1$  for all  $j$  require that 100% of customer  $j$  demand is satisfied. The constraint  $-x_{ij} + y_i \geq 0$  “forces” plant  $i$  to be open if it supplies any of the demand at location  $j$ .

This model is called the simple or uncapacitated plant location problem because an open plant can supply 100% of demand at every customer location. There is a wide literature on this problem. Model formulations go back at least as far as Balinski [31]. Efronymson and Ray [138] give an enumerative algorithm for this problem where the constraints  $-x_{ij} + y_i \geq 0$  are replaced by aggregate constraints of the form  $-\sum_{j=1}^m x_{ij} + my_i \geq 0$ . Benders’ decomposition approaches are discussed in Balinski [32], Doherty [126], Guignard and Spielberg [217], and Spielberg [410]. Recall that in the lock box model the  $y_i$  variable is used to flag whether or not lock box  $i$  is open. The variable  $x_{ij} = 1$  if customer  $j$  is assigned to lock box  $i$ . The cost  $c_{ij}$  is an annual opportunity cost of assigning customer  $j$  to lock box  $i$  and  $f_i$  is the annual fixed cost of operating lock box  $i$ . See Cornuéjols, Fisher, and Nemhauser [101].

At each iteration, the subproblem  $\max\{(b - B\bar{y})^\top u \mid A^\top u \leq c, u \geq 0\}$  is solved for  $u$ . The dual of the subproblem is  $\min\{c^\top x + f^\top y \mid Ax \geq b - B\bar{y}, x \geq 0\}$ , which is the original problem with  $y$  fixed at  $\bar{y}$  and  $u$  is the vector of multipliers on the constraints  $Ax \geq b - B\bar{y}$ . For the simple plant location problem  $(SPL)$ , let  $O(\bar{y}) := \{i \mid \bar{y}_i = 1\}$  index the open plants and  $C(\bar{y}) := \{i \mid \bar{y}_i = 0\}$  index

the closed plants. The Benders' subproblem is

$$(SPL(\bar{y})) \quad \begin{aligned} \text{min } & \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} + \sum_{i \in O(\bar{y})} f_i \\ \text{s.t. } & \sum_{i=1}^n x_{ij} \geq 1, \quad j = 1, \dots, m \\ & x_{ij} \leq 1, \quad i \in O(\bar{y}), \quad j = 1, \dots, m \\ & x_{ij} \leq 0, \quad i \in C(\bar{y}), \quad j = 1, \dots, m \\ & x_{ij} \geq 0, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \end{aligned}$$

We illustrate the Benders' algorithm on the simple plant location model in the Example 10.6.

**Example 10.6** Assume there are three potential plants and five customers. The relevant costs are in Table 10.1.

Table 10.1 Simple Plant Location Data for Example 10.6

	CUSTOMER					FIXED COSTS
	1	2	3	4	5	
PLANT	1	2	3	4	5	7
2	4	3	1	2	6	3
3	5	4	2	1	3	3

Relating this example back to the generic formulation (10.1)-(10.4) gives

$$A = \left[ \begin{array}{cccccc|cccccc|cccccc} 1 & & & & 1 & & & 1 & & & 1 & & & 1 & & & \\ & 1 & & & & 1 & & & & 1 & & & & 1 & & & \\ & & 1 & & & & 1 & & & & & 1 & & & & 1 & & \\ & & & 1 & & & & 1 & & & & & 1 & & & & 1 & \\ & & & & 1 & & & & & 1 & & & & & 1 & & & \\ -1 & & & & & -1 & & & & & -1 & & & & & -1 & & & \\ & -1 & & & & & -1 & & & & & -1 & & & & & -1 & & \\ & & -1 & & & & & -1 & & & & & -1 & & & & & -1 & \\ & & & -1 & & & & & -1 & & & & & -1 & & & & -1 & \\ & & & & -1 & & & & & -1 & & & & & -1 & & & -1 & \\ & & & & & -1 & & & & & -1 & & & & & -1 & & & -1 \end{array} \right]$$

The binary constraint  $y_i \in \{0, 1\}$  corresponds to the set  $Y$ .

### Iteration 1:

Start with an initial  $y \in Y$ . It is trivial to find such an initial feasible solution to this problem. One feasible solution is  $\bar{y}_1 = 1$ ,  $\bar{y}_2 = 0$ ,  $\bar{y}_3 = 0$ . The resulting subproblem is

$$\begin{aligned}
& \min 2x_{11} + 3x_{12} + 4x_{13} + 5x_{14} + 7x_{15} \\
& + 4x_{21} + 3x_{22} + x_{23} + 2x_{24} + 6x_{25} \\
& + 5x_{31} + 4x_{32} + 2x_{33} + x_{34} + 3x_{35} + 2 \\
x_{11} + x_{21} + x_{31} & \geq 1 \\
x_{12} + x_{22} + x_{32} & \geq 1 \\
x_{13} + x_{23} + x_{33} & \geq 1 \\
x_{14} + x_{24} + x_{34} & \geq 1 \\
x_{15} + x_{25} + x_{35} & \geq 1
\end{aligned}$$

$$\begin{aligned}
x_{11} &\leq 1, x_{21} \leq 0, x_{31} \leq 0 \\
x_{12} &\leq 1, x_{22} \leq 0, x_{32} \leq 0 \\
x_{13} &\leq 1, x_{23} \leq 0, x_{33} \leq 0 \\
x_{14} &\leq 1, x_{24} \leq 0, x_{34} \leq 0 \\
x_{15} &\leq 1, x_{25} \leq 0, x_{35} \leq 0 \\
x_{ij} &\geq 0, i = 1, \dots, 3, j = 1, \dots, 5.
\end{aligned}$$

The solution to this subproblem (which is the dual of what we are calling the Benders' subproblem) is  $x_{11} = x_{12} = x_{13} = x_{14} = x_{15} = 1$ . This gives an upper bound  $UB = [2 + 3 + 4 + 5 + 7] + 2 = 23$ . The lower bound is  $-\infty$  by default. Since the upper and lower bounds are not equal it is necessary to find an optimal set of dual variables from the current subproblem and generate a cut for the master.

For the Benders' decomposition algorithm to be effective it is essential that the linear programming subproblem have special structure so that it is easily optimized, preferably without using a simplex or barrier algorithm. To make solution for the optimal  $u$  more transparent, let  $u = (v, w)$  where  $v$  is a vector of dual variables associated with the demand constraints, and  $w$  a vector of dual variables associated with the setup constraints. Based on the structure of the subproblem for the simple plant location problem it is easy to find optimal  $v_j$  and  $w_{ij}$ . In the simple plant location problem, once the plant locations are decided (i.e., which  $y_i$  are fixed to one) the allocation decision is easy. Since there are no capacity constraints, all of the demand at location  $j$  is satisfied from the closest (i.e. cheapest) open plant. Therefore, if the right hand side of the  $j$ th demand constraint  $\sum_{i=1}^3 x_{ij} \geq 1$  is decreased by a small amount  $\epsilon$ , we save  $\epsilon$  times the cost of satisfying demand from the closest open plant. That is,  $v_j = \min_{i \in O(\bar{y})} \{c_{ij}\}$  for all  $j$ . Now consider dual variable  $w_{ij}$  on the constraint  $-x_{ij} \geq -\bar{y}_i$ . If plant  $i$  is open, this constraint is  $-x_{ij} \geq -1$  and decreasing the right hand side by  $\epsilon$  gives  $-x_{ij} \geq -1 - \epsilon$  or  $x_{ij} \leq 1 + \epsilon$ . Because demand constraint  $j$  has a right hand side of 1 the "extra" supply of  $\epsilon$  on the constraint  $x_{ij} \leq 1 + \epsilon$  does nothing. Therefore,  $w_{ij} = 0$  for all  $i \in O(\bar{y})$ . However, if plant  $i$  is closed, i.e.  $-x_{ij} \geq 0$  then decreasing the right hand side by  $\epsilon$  so that the constraint is now  $-x_{ij} \geq -\epsilon$  may help. We have increased the supply from 0 to  $\epsilon$  so if plant  $i$  is closed, but is closer to customer  $j$  than any of the open plants, we save money. That is, we save  $\epsilon(v_j - c_{ij})$  where  $v_j$  is the current cost of serving customer  $j$ . This logic implies

$$w_{ij} = 0, i \in O(\bar{y}), w_{ij} = \max\{(v_j - c_{ij}), 0\}, i \in C(\bar{y}).$$

This intuitive result is stated formally in Proposition 10.7. A rigorous proof is left as Exercise 10.3.

**Proposition 10.7** *An optimal set of dual variables to the simple plant location Benders' subproblem ( $SPL(\bar{y})$ ) is*

$$v_j = \min_{i \in O(\bar{y})} \{c_{ij}\} \quad j = 1, \dots, m$$

$$w_{ij} = 0, \quad i \in O(\bar{y}), \quad w_{ij} = \max\{(v_j - c_{ij}), 0\}, \quad i \in C(\bar{y}).$$

**Example 10.8 (Example 10.6 continued.)** *For this example problem Proposition 10.7 gives*

$$\begin{aligned} v_1 &= 2, \quad v_2 = 3, \quad v_3 = 4, \quad v_4 = 5, \quad v_5 = 7, \\ w_{11} &= w_{12} = w_{13} = w_{14} = w_{15} = 0, \\ w_{21} &= 0, \quad w_{22} = 0, \quad w_{23} = 3, \quad w_{24} = 3, \quad w_{25} = 1, \\ w_{31} &= 0, \quad w_{32} = 0, \quad w_{33} = 2, \quad w_{34} = 4, \quad w_{35} = 4. \end{aligned}$$

Add the corresponding Benders' cut to the master. The cut is

$$z_0 \geq u^\top (b - By) + f^\top y$$

where  $u = [v, w]$ . Recall that the only nonzeros of  $b$  are in the demand constraints so the  $w_{ij}$  are multiplied by 0 and  $u^\top b = \sum_{j=1}^5 v_j$ . Refer back to the structure of the  $B$  matrix and observe that the only nonzero elements occur in the setup forcing constraints so the  $v_j$  are multiplied by 0. Then  $u^\top B = \sum_{i=1}^3 \sum_{j=1}^5 w_{ij}$ . Define  $\rho_i = \sum_{j=1}^5 w_{ij}$ . Then  $z_0 \geq u^\top (b - By) + f^\top y$  is equivalent to

$$z_0 \geq \sum_{j=1}^5 v_j + \sum_{i=1}^3 (f_i - \rho_i)y_i.$$

The first master program is

$$\begin{array}{ll} \min & z_0 \\ \text{s.t.} & z_0 \geq 21 + 2y_1 - 4y_2 - 7y_3 \\ & y_1 + y_2 + y_3 \geq 1 \\ & y_i \in \{0, 1\} \end{array}$$

The optimal solution to this problem is  $y_1 = 0$ ,  $y_2 = 1$ ,  $y_3 = 1$  for a  $z_0 = 10$ . Thus  $UB = 23$ ,  $LB = 10$ . What is the purpose of the constraint  $y_1 + y_2 + y_3 \geq 1$ ?

**Second Iteration:**

With plants 2 and 3 open the subproblem solution is

$$\begin{aligned}x_{21} &= 1, x_{22} = 1, x_{23} = 1, x_{34} = 1, x_{35} = 1 \\UB &= [4 + 3 + 1 + 1 + 3] + 6 = 18\end{aligned}$$

$$v_1 = 4, v_2 = 3, v_3 = 1, v_4 = 1, v_5 = 3$$

$$\begin{aligned}w_{21} &= w_{22} = w_{23} = w_{24} = w_{25} = 0 \\w_{31} &= w_{32} = w_{33} = w_{34} = w_{35} = 0\end{aligned}$$

$$w_{11} = 2, w_{12} = 0, w_{13} = 0, w_{14} = 0, w_{15} = 0$$

The second master program is

$$\begin{array}{ll}\min & z_0 \\ \text{s.t.} & z_0 \geq 21 + 2y_1 - 4y_2 - 7y_3 \\ & z_0 \geq 12 + 0y_1 + 3y_2 + 3y_3 \\ & y_1 + y_2 + y_3 \geq 1 \\ & y_1, y_2, y_3, \in \{0, 1\}\end{array}$$

The optimal solution is  $y_1 = 0, y_2 = 0, y_3 = 1, z_0 = 15$  and a  $UB = 18, LB = 15$ . At the second iteration the lower bound has increased from 10 to 15 and the upper bound decreased from 23 to 18.

### Third Iteration:

With plant 3 open the resulting LP solution is

$$\begin{aligned}x_{31} &= 1, x_{32} = 1, x_{33} = 1, x_{34} = 1, x_{35} = 1 \\UB &= [5 + 4 + 2 + 1 + 3] + 3 = 18 \text{ no improvement} \\v_1 &= 5, v_2 = 4, v_3 = 2, v_4 = 1, v_5 = 3\end{aligned}$$

$$\begin{aligned}w_{31} &= w_{32} = w_{33} = w_{34} = w_{35} = 0 \\w_{11} &= 3, w_{12} = 1, w_{13} = 0, w_{14} = 0, w_{15} = 0 \\w_{21} &= 1, w_{22} = 1, w_{23} = 1, w_{24} = 0, w_{25} = 0\end{aligned}$$

The third master program is

$$\begin{array}{ll}\min & z_0 \\ \text{s.t.} & z_0 \geq 21 + 2y_1 - 4y_2 - 7y_3 \\ & z_0 \geq 12 + 0y_1 + 3y_2 + 3y_3 \\ & z_0 \geq 15 - 2y_1 + 0y_2 + 3y_3 \\ & y_1 + y_2 + y_3 \geq 1 \\ & y_1, y_2, y_3, \in \{0, 1\}\end{array}$$

The optimal solution  $y_1 = 1, y_2 = 0, y_3 = 1, z_0 = 16$ . At this iteration  $UB - LB = 18 - 16 = 2$  and the process continues in an identical fashion.

## 10.4 DUAL VARIABLE SELECTION

Location, distribution and transportation models often have optimal linear programming solutions which are highly primal degenerate. Although primal degeneracy does not necessarily imply dual alternative optima, see the discussion in Section 2.8, these models typically have alternative dual optima. In this section we show that the selection of the dual variables affects the quality of Benders' cuts.

**Example 10.9 (Example 10.6 Continued)** In Example 10.6 the solution  $\bar{y}_1 = 1, \bar{y}_2 = \bar{y}_3 = 0$  yielded the cut  $z_0 \geq 21 + 2y_1 - 4y_2 - 7y_3$ . Recall the Benders' cut is

$$z_0 \geq \sum_{j=1}^5 v_j + \sum_{i=1}^3 (f_i - \rho_i)y_i$$

where,

$$\rho_i = \sum_{j=1}^5 w_{ij} = \sum_{j=1}^5 \max\{0, v_j - c_{ij}\}.$$

In Proposition 10.7 the value of  $v_j$  was based on reducing the right hand side of the demand constraint at location  $j$  from 1 to  $1 - \epsilon$ . However, what if this right hand side were changed to  $1 + \epsilon$ ? If  $k$  is closest open plant to location  $j$  there is a constraint  $x_{kj} \leq 1$  in the subproblem. Therefore, it is not possible to set  $x_{kj} = 1 + \epsilon$  and location  $j$  must get the additional  $\epsilon$  from another plant where  $c_{ij} \geq c_{kj}$ . Let  $k$  be an open plant such that

$$c_{kj} = \min\{c_{ij} \mid i \in O(\bar{y})\},$$

and define,  $v_j = \min\{c_{ij} \mid i \neq k \text{ and } c_{ij} \geq c_{kj}\}$ . If there are no  $i \neq k$  such that  $c_{ij} \geq c_{kj}$ , then  $v_j = c_{kj}$ . As before, let  $w_{ij} = \max\{0, v_j - c_{ij}\}$ . This gives  $v_1 = 4, v_2 = 3, v_3 = 4, v_4 = 5, v_5 = 7$  and we have

$$\begin{aligned} z_0 &\geq 23 + 0y_1 - 4y_2 - 7y_3 && \text{New Cut} \\ z_0 &\geq 21 + 2y_1 - 4y_2 - 7y_3 && \text{Original Cut} \end{aligned}$$

The new cut can never do worse than the original cut. Regardless of the values of  $y_i$ , the new cut will always "force"  $z_0$  to as large, if not larger, value than

that implied by the original cut. This example and the following Proposition raise several interesting questions.

**Proposition 10.10** *Let  $\bar{v}_j = c_{kj} = \min \{c_{ij} \mid i \in O(\bar{y})\}$  for all  $j$ . Let  $L_j := \min \{c_{ij} \mid i \in O(\bar{y}), i \neq k \text{ and } c_{ij} > c_{kj}\}$ . If there are no  $i \neq k, i \in O(\bar{y})$  such that  $c_{ij} > c_{kj}$  then  $L_j := c_{kj}$ . If  $\bar{v}_j \leq v_j \leq L_j$  for all  $j$  and  $w_{ij} = \max \{0, v_j - c_{ij}\}$  then  $v_j, w_{ij}$  constitute an optimal set of dual variables to the primal simple plant location subproblem ( $SPL(\bar{y})$ ).*

Given Proposition 10.10, how large should we make the  $v_j$ ? This is *not easy* to answer. We just saw an example where increasing the  $v_j$  slightly could never hurt. But consider the following example.

**Example 10.11 (Example 10.6 Continued)** Fix  $\bar{y}_1 = 0, \bar{y}_2 = 1, \bar{y}_3 = 0$ . Using Proposition 10.7 the dual variables are  $v_1 = 4, v_2 = 3, v_3 = 1, v_4 = 2, v_5 = 6$ . The resulting cut is

$$z_0^1 \geq 16 + 0y_1 + 3y_2 - y_3.$$

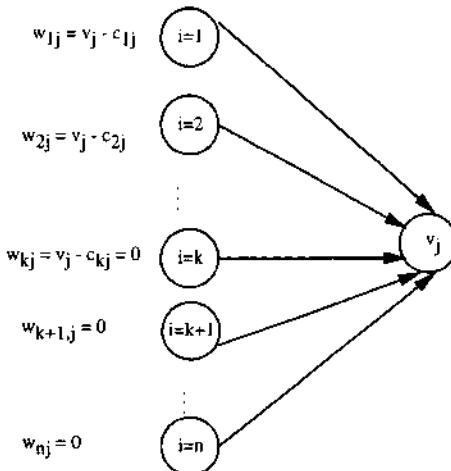
An alternative dual variables using Proposition 10.10 are  $v_1 = 5, v_2 = 3, v_3 = 2, v_4 = 5, v_5 = 7$ . This gives the cut

$$z_0^2 \geq 22 - y_1 - 3y_2 - 5y_3.$$

If  $y_1 = y_2 = y_3 = 1$  then  $z_0^1 \geq 18$  and  $z_0^2 \geq 13$ . For this solution the first cut is superior, however if  $y_1 = 1, y_2 = y_3 = 0$  then  $z_0^1 \geq 16, z_0^2 \geq 21$  and the second cut appears superior.

How do we decide upon which cut we want? Magnanti and Wong [305] have taken an important step in answering this question. Cut  $z_0 \geq f^\top y + (u^1)^\top (b - By)$  dominates or is stronger than cut,  $z_0 \geq f^\top y + (u^2)^\top (b - By)$  if and only if  $f^\top y + (u^1)^\top (b - By) \geq f^\top y + (u^2)^\top (b - By)$  for all  $y \in Y$  and there exists at least one  $y \in Y$  such that strict inequality holds. For example, the cut  $z_0 \geq 23 + 0y_1 - 4y_2 - 7y_3$  dominates the cut  $z_0 \geq 21 + 2y_1 - 4y_2 - 7y_3$ . A cut is Pareto optimal if it is not dominated by any other cut.

The following Figure is useful in understanding how to generate Pareto optimal cuts for simple plant location.

**Figure 10.1** Dual Variable Selection

In Figure 10.1 assume without loss that the plant indices are ordered so that  $c_{i+1,j} \geq c_{ij}$ . Plant  $k$  indexes the open plant which is closest to demand location  $j$ . Now refer to Example 10.9 and consider the case of  $v_1$ . The closest open plant is plant one, so  $k = 1$ . What is the effect on the cut  $z_0 \geq \sum_{j=1}^5 v_j + \sum_{i=1}^3 (f_i - \rho_i)y_i$  from increasing  $v_1$  from 2 to 4? Increasing  $v_1$  from 2 to 4 increases the constant term  $\sum_{j=1}^5 v_j$  by 2 units but also has the effect increasing  $w_{11}$  from 0 to 2. This increases  $\rho_1$  by 2 units. The net effect is that for any setting of the  $y_i$  with  $y_1 = 0$ ,  $z_0$  is increased by 2 units. If  $y_1 = 1$  the net effect is zero. Clearly it can never hurt, and will always help when  $y_1 = 0$ , to increase  $v_1$  from 2 to 4. Now consider what happens in Example 10.11. In this case,  $k = 2$  and using Proposition 10.7,  $v_1 = 4$ . Using Proposition 10.10 we can increase  $v_1$  to 5. The result of this increase is to increase the constant term  $\sum_{j=1}^5 v_j$  by 1 unit. However, unlike Example 10.9, in this example  $k = 2$ , so both  $w_{11}$  and  $w_{21}$  increase by 1. Then both  $\rho_1$  and  $\rho_2$  increase by one unit. This means that setting  $y_1 = 1$  and  $y_2 = 1$  will hurt the bound on  $z_0$ . As illustrated in Figure 10.1, whenever  $v_j$  is increased above the value  $c_{kj}$  the  $w_{ij}$  dual variable will also increase for all  $i \leq k$ . This suggests the following method for selecting dual variables and gives a Pareto optimal cut.

Let  $I_j = \{i \mid c_{ij} \leq c_{kj}, i \neq k\}$ . Define a set of dual variables by

$$v'_j = \begin{cases} \min\{c_{ij} \mid i \in O(\bar{y})\}, & \text{if } I_j \neq \emptyset \\ \min\{c_{ij} \mid i \neq k\}, & \text{if } I_j = \emptyset \end{cases} \quad (10.9)$$

$$w'_{ij} = \max\{0, v'_j - c_{ij}\}. \quad (10.10)$$

Apparently Balinski [32] was the first to realize the value of dual variables of this type although he did not express them in quite this fashion.

**Proposition 10.12 (Balinski)** *The dual variables defined by (10.9)-(10.10) will yield a cut which is Pareto optimal.*

**Example 10.13 (Example 10.6 Continued)** *Let  $\bar{y}_1 = 0, \bar{y}_2 = 1, \bar{y}_3 = 0$ . We get  $v'_1 = 4, v'_2 = 3, v'_3 = 2, v'_4 = 2, v'_5 = 6$ . The corresponding cut is*

$$z_0 \geq 17 + 0y_1 + 2y_2 - y_3$$

The key issue in this discussion is the cardinality of set  $I_j$ . If  $|I_j| = N$  then increasing  $v_j$  above the value  $c_{kj}$ , cannot hurt the bound on  $z_0$  only if  $y_i = 0$  for all  $i \in I_j$ . If  $Y = \{(y_1, \dots, y_n) | y_i \in \{0, 1\}\}$  then at most  $\frac{1}{2^N}$  of all possible solutions will give a better bound on  $z_0$ . Thus, as  $N$  gets larger only a very small fraction of the integer solution make increasing  $v_j$  an effective procedure. Of course, we may increase several  $v_j$  which makes the issue more complicated.

**Example 10.14 (Example 10.6 Continued)** *Consider the following possible cuts for  $\bar{y}_1 = 0, \bar{y}_2 = 1, \bar{y}_3 = 0$ .*

Cut #1 (Proposition 10.7),  $v_1 = 4, v_2 = 3, v_3 = 1, v_4 = 2, v_5 = 6$

$$z_0 \geq 16 + 0y_1 + 3y_2 - y_3$$

Cut #2 (Proposition 10.10),  $v_1 = 5, v_2 = 3, v_3 = 2, v_4 = 5, v_5 = 7$

$$z_0 \geq 22 - y_1 - 3y_2 - 5y_3 \quad (\text{We increased } v_1 \text{ to 5 from 4, } v_3 \text{ to 2 from 1, } v_4 \text{ to 5 from 2 and } v_5 \text{ to 7 from 6})$$

Cut #3 (Proposition 10.12),  $v_1 = 4, v_2 = 3, v_3 = 2, v_4 = 2, v_5 = 6$

$$z_0 \geq 17 + 0y_1 + 2y_2 - y_3 \quad (\text{We increased } v_3 \text{ to 2 from 1})$$

*The bound generated for each of the three cuts, for each integer solution, is given in Table 10.2.*

**Table 10.2** Cut Bounds Generated for Different Dual Solutions

Possible Solutions	Cut 1	Cut 2	Cut 3
$y_1 = 0 \quad y_2 = 0 \quad y_3 = 0$	16	22	17
$y_1 = 1 \quad y_2 = 1 \quad y_3 = 1$	18	13	18
$y_1 = 1 \quad y_2 = 1 \quad y_3 = 0$	19	18	19
$y_1 = 1 \quad y_2 = 0 \quad y_3 = 0$	16	21	17
$y_1 = 1 \quad y_2 = 0 \quad y_3 = 1$	15	16	16
$y_1 = 0 \quad y_2 = 1 \quad y_3 = 1$	18	14	18
$y_1 = 0 \quad y_2 = 0 \quad y_3 = 1$	15	17	16
$y_1 = 0 \quad y_2 = 1 \quad y_3 = 0$	19	19	19

We are always better off going with a cut defined by (10.9)-(10.10) instead of the standard dual variables. This is so because we can increase a  $v_j$  while experiencing a decrease in a single  $w_{ij}$ . This cut is also Pareto optimal. It doesn't appear any other definitive statements can be made about how the  $v_j$  should be increased. One should be aware that as the  $v_j$  are increased above the values given by (10.9)-(10.10) at least two  $\rho_i$  will be decreasing per unit of  $v_j$ . It might be desirable to add several cuts at each iteration. We could add the cut given by (10.9)-(10.10) and then add the cut corresponding to the other extreme. That is, define the  $v_j$  by  $L_j$ .

## 10.5 CONCLUSION

Perhaps the most successful implementation of Benders' decomposition is described in Geoffrion and Graves [175]. Benders' decomposition was used to solve a model for finding distribution center locations for Hunt-Wesson Foods, Inc. Doherty [126] describes the application of Benders' decomposition to several location problems at the Procter & Gamble company. Bahl and Zions [19] apply Benders' decomposition to multiproduct lot sizing. The key feature of all these applications is that the subproblem constraints  $Ax \geq b - B\bar{y}$  have *special structure*. Unless the subproblems can be solved very efficiently (i.e. without resorting to general linear programming codes) it is probably best to apply another method of decomposition or to use linear programming based branch and bound.

A number of authors have worked on extensions and improvements of Benders' decomposition. Geoffrion [173] has extended the basic algorithm to nonlinear problems if the set  $Y$  in constraint (10.3) is a convex set. Shapiro [406] ap-

plies Benders' decomposition to the node aggregation problem. McDaniel and Devine [321] present an algorithm for accelerating the basic Benders' algorithm by generating cuts based on the linear programming relaxation. Finally, as discussed in the previous section, Magnanti and Wong [305] address the important issue of dual variable selection.

Later, in Chapter 16, Benders' decomposition is used once again. It is used in Chapter 16 to solve integer programming models that have been reformulated using auxiliary variables. The auxiliary variables are projected out using a Benders' like algorithm to get valid cutting planes in the space of the original variables.

## 10.6 EXERCISES

- 10.1 In the simplex algorithm it is not unusual to have degenerate pivots. That is, bringing in a new column has no effect on the solution value. Now consider Benders' decomposition. Assume that adding a cut to the relaxed master results in *no change* in the objective function value. What does this imply?
- 10.2 If the Benders' subproblem is infeasible what does this imply about the original problem?
- 10.3 Prove Proposition 10.7.
- 10.4 Finish the simple plant location example developed in Section 10.3.
- 10.5 Prove Proposition 10.10.
- 10.6 Refer to Example 12.20. Take the data for the capacitated lot sizing problem and perform three iterations of Benders decomposition.
- 10.7 Can the same Benders' cut be generated at two different iterations? If so, what is the implication of this?
- 10.8 The Sender company is in the process of planning for new production facilities and developing a more efficient distribution system design. At present they have one plant in St. Louis with a capacity of 30,000 units. Because of increased demand, management is considering four potential new plant sites: Detroit, Denver, Toledo, and Kansas City. The transportation tableau below summarizes the projected plant capacities, the cost per unit of shipping from each plant to each customer zone, and the

demand forecasts over a one-year planning horizon. The fixed costs of constructing the new plants are: Detroit \$175,000, Toledo \$300,000, Denver \$375,000, and Kansas City \$500,000.

- a. Formulate this as a mixed 0/1 linear program.
- b. Add a constraint which will guarantee that the solution of the Benders' relaxed master is feasible for the subproblem.
- c. Perform three iterations of Benders' decomposition on this problem. Solve the primal subproblem, not its dual. Begin with the solution which has every plant open.
- d. Simplex codes give the dual variables as part of the output. Assume they did not. Clearly describe how you could obtain the optimal dual variables given the optimal primal solution. The algorithm should be somewhat similar to what we do in the uncapacitated case – do not give a procedure which involves any matrix inversions, etc.

Plant	Zone 1	Zone 2	Zone 3	Supply
Detroit	5	2	3	10,000
Toledo	4	3	4	20,000
Denver	9	7	5	30,000
Kansas	10	4	2	40,000
St. Louis	8	4	3	30,000
Demand	30,000	20,000	20,000	

10.9 Solve the segregated storage problem from Exercise 9.16 using Benders' decomposition (do at least three iterations). Use an integer linear programming package to solve the restricted master programs, but do not use a linear programming package to solve the subproblems. Show how to optimize the subproblems without using simplex or an interior point algorithm.

10.10 In the simple plant location problem the constraints  $x_{ij} \leq y_i$ , for all  $i, j$  can be replaced by the aggregate constraints  $\sum_{j=1}^m x_{ij} \leq my_i$  for all  $i$ . Thus,  $nm$  constraints are replaced with  $n$  constraints which represents a considerable savings. Convince yourself that for any fixed  $y$  vector the formulations are equivalent. Therefore, it does not matter which formulation is used with Benders' decomposition. Either support or refute this statement.

10.11 Prove Proposition 10.12.

- 10.12 Refer to Proposition 10.10. Is it possible to obtain an optimal set of dual variables for the simple plant location with  $v_j > L_j$ ?
- 10.13 In the formulation (10.1)-(10.4) assume that  $Y = \{(y_1, \dots, y_n) | y_i \in \{0, 1\}\}$ . If  $\bar{u}$  are the optimal dual variables on the linear programming relaxation, defined by replacing  $y \in Y$  with  $0 \leq y_i \leq 1$ , what is the interpretation of  $d - B^T \bar{u}$ ?

# INVERSE PROJECTION: DANTZIG-WOLFE DECOMPOSITION

## 11.1 INTRODUCTION

Just as inverse projection is the dual of projection, *Dantzig-Wolfe decomposition*, see Dantzig and Wolfe [112], is the dual of Benders' decomposition. With Dantzig-Wolfe decomposition, the linear program is decomposed into two sets of constraints, whereas in Benders' decomposition, the linear program was decomposed into two sets of variables. Consider the linear program (*LP*)

$$\begin{aligned} (LP) \quad & \min c^\top x \\ & \text{s.t. } Ax \geq b \\ & \quad Bx \geq d \\ & \quad x \geq 0, \end{aligned}$$

where  $c$ ,  $x$ ,  $b$ , and  $d$  are vectors,  $A$  and  $B$  are matrices, all of conformable dimension. Instead of projecting out variables we do the dual operation and replace (inverse project) the constraints  $Bx \geq d$  with variables. Of course this may result in a linear program with far too many variables to solve. Rather than solve the linear program with all the variables present, the variables are added on the fly as needed. Note the complete constraint – variable duality here with Benders' decomposition.

To take full advantage of this decomposition procedure one should partition the constraint matrix into  $A$  and  $B$  such that the  $B$  matrix 1) contains the *vast majority* of the constraints, and 2) contains *special structure* that makes the linear program over the  $Bx \geq d$  constraints significantly easier to optimize than the original problem. In the next section the Dantzig-Wolfe decomposition algorithm is developed. In Section 11.3 we again illustrate our methods

with a simple plant location example. In Section 11.4 we further specialize Dantzig-Wolfe to the case where the constraint set  $Bx \geq d$  decomposes into a set of disjoint block constraints. In Section 11.5 there is a discussion of the computational aspects of Dantzig-Wolfe decomposition. Concluding remarks are given in Section 11.6. Exercises are provided in Section 11.7.

## 11.2 DANTZIG-WOLFE DECOMPOSITION

Dantzig-Wolfe decomposition is an application of inverse projection to linear programs with special structure. Benders' decomposition is based on the result that projection yields all of the extreme rays of the projection cone, Dantzig-Wolfe decomposition is based on the dual result that inverse projection provides all of the generators of a polytope. It is based on Theorem 3.3 repeated here for completeness.

**Theorem 11.1 (Finite Basis Theorem for Polyhedra (Minkowski))** *If  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ , then  $P$  is finitely generated. In particular, there exist  $x^1, \dots, x^q, x^{q+1}, \dots, x^r$  in  $\mathbb{R}^n$  such that*

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}).$$

*Furthermore, the extreme points of  $P$  are contained in the set  $\{x^1, \dots, x^q\}$  and the extreme rays of the recession cone  $\{x \in \mathbb{R}^n \mid Ax = 0, x \geq 0\}$  are contained in the set  $\{x^{q+1}, \dots, x^r\}$ .*

**Key Idea One - An Equivalent Master Problem.** For simplicity, assume that  $\{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0\}$  is a polytope. Then any point in this polytope can be written as a convex combination of the extreme points of the polytope. Let  $x^1, \dots, x^q$  be the extreme points of this polytope. If  $\bar{x}$  is a point in the polytope  $\{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0\}$ , then there are  $\bar{z}_1, \dots, \bar{z}_q$  such that

$$\bar{x} = \sum_{i=1}^q \bar{z}_i x^i, \quad \sum_{i=1}^q \bar{z}_i = 1, \quad \bar{z}_i \geq 0, \quad i = 1, \dots, q.$$

Therefore, an equivalent problem to the original problem ( $LP$ ) is

$$\begin{array}{ll} \min & c^\top x \\ \text{s.t.} & Ax \geq b \end{array}$$

$$\begin{aligned} \sum_{i=1}^q z_i x^i &= x \\ \sum_{i=1}^q z_i &= 1 \\ z &\geq 0. \end{aligned}$$

This is a problem with both  $x$  and  $z$  variables. Next substitute  $x = \sum_{i=1}^q z_i x^i$  into the constraint set  $Ax \geq b$  and into the objective function  $c^\top x$ . The resulting linear program is

$$\begin{aligned} \min \quad & c^\top \left( \sum_{i=1}^q z_i x^i \right) \\ \text{s.t.} \quad & A \left( \sum_{i=1}^q z_i x^i \right) \geq b \\ & \sum_{i=1}^q z_i = 1 \\ & z \geq 0. \end{aligned}$$

This is simplified by observing

$$c^\top \left( \sum_{i=1}^q z_i x^i \right) = \sum_{i=1}^q (c^\top x^i) z_i \quad \text{and} \quad A \left( \sum_{i=1}^q z_i x^i \right) = \sum_{i=1}^q (Ax^i) z_i,$$

and defining  $f_i := c^\top x^i$ ,  $p^i := Ax^i$ ,  $i = 1, \dots, q$ . Then the original problem ( $LP$ ) is equivalent to

$$\min \quad \sum_{i=1}^q f_i z_i \tag{11.1}$$

$$(DW) \quad \text{s.t.} \quad \sum_{i=1}^q p^i z_i \geq b \tag{11.2}$$

$$\sum_{i=1}^q z_i = 1 \tag{11.3}$$

$$z \geq 0. \tag{11.4}$$

**Example 11.2** Consider the linear program

$$\min \quad -x_1 - x_2$$

$$\begin{array}{ll} \text{s.t.} & x_1 - x_2 \leq 2 \\ & 4x_1 + 9x_2 \leq 18 \\ & -2x_1 + 4x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{array}$$

Treat the first constraint as the constraint set  $Ax \geq b$  and the second and third constraints as the constraint set  $B \geq d$ . The theoretical development so far does not depend in any way on the constraints of the linear program having any special form, i.e. equality, less than or equal, or greater than or equal.

There are four extreme points in the polytope defined by the second and third constraints, along with the nonnegativity constraints. The coordinates of these four points are  $(0, 0)$ ,  $(4(1/2), 0)$ ,  $(1(1/17), 1(9/17))$  and  $(0, 1)$ . Any point in this polytope can be written as

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} z_1 + \begin{bmatrix} 4(1/2) \\ 0 \end{bmatrix} z_2 + \begin{bmatrix} 1(1/17) \\ 1(9/17) \end{bmatrix} z_3 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} z_4$$

where  $z_1 + z_2 + z_3 + z_4 = 1$  and are nonnegative.

Since  $A = [1, -1]$ , the  $p^i = Ax^i$ , for  $i = 1, 2, 3, 4$  are

$$p^1 = [1, -1] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = [0], p^2 = [1, -1] \begin{bmatrix} 4(1/2) \\ 0 \end{bmatrix} = [4(1/2)],$$

$$p^3 = [1, -1] \begin{bmatrix} 1(1/17) \\ 1(9/17) \end{bmatrix} = [-(8/17)], p^4 = [1, -1] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [-1].$$

The objective function coefficients in the master program are  $c^T x^i = [-1, -1] x^i$ . This gives

$$f_1 = [-1, -1] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = [0], f_2 = [-1, -1] \begin{bmatrix} 4(1/2) \\ 0 \end{bmatrix} = [-4(1/2)],$$

$$f_3 = [-1, -1] \begin{bmatrix} 1(1/17) \\ 1(9/17) \end{bmatrix} = \begin{bmatrix} -2(10/17) \end{bmatrix}, f_4 = [-1, -1] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix}.$$

The master program is

$$\begin{aligned} \min \quad & 0z_1 - 4(1/2)z_2 - 2(10/17)z_3 - z_4 \\ \text{s.t.} \quad & 0z_1 + 4(1/2)z_2 - (8/17)z_3 - z_4 \leq 2 \\ & z_1 + z_2 + z_3 + z_4 = 1 \\ & z_1, z_2, z_3, z_4 \geq 0 \end{aligned}$$

The optimal solution to the master is  $z_1 = 0.497$ ,  $z_2 = 0.503$  for an objective function value of  $-3.538$ . What are the corresponding optimal  $x_1$  and  $x_2$ ? Is this optimal solution an extreme point of the polytope defined by the second and third constraints along with the nonnegativity constraints?

The linear program (DW) defined in (11.1)-(11.4) is the *Dantzig-Wolfe master program*. Constraints (11.2) are the *coupling constraints* and constraint (11.3) is a *convexity row*. It is also an example of a generalized upper bound (GUB) constraint, or a type 1 special ordered set. If  $A$  and  $B$  have  $m_1, m_2$  rows respectively, then the original problem formulation has  $m_1 + m_2$  rows. However, the Dantzig Wolfe master has only  $m_1 + 1$  constraints. This is why we want to partition the original constraint matrix into  $A$  and  $B$  so that  $B$  contains the vast majority of rows in the problem. (You will see shortly why it is pointless to put *all* the rows into  $B$ .) Unfortunately, there are *many* more variables in the Dantzig-Wolfe master than in the original formulation since we create a variable for every extreme point in the *subproblem polytope*  $\{x \mid Bx \geq d, x \geq 0\}$ . Regardless of how cheap hard disk memory has become, it is not practical to store the coefficients for all the extreme points of a realistic problem. This motivates the second key idea.

**Key Idea Two - Column Generation.** The revised simplex algorithm is applied to a *restricted master program (DWR)* which is

$$\min \sum_{i \in \Lambda} f_i z_i \tag{11.5}$$

$$(DWR) \quad \text{s.t.} \quad \sum_{i \in \Lambda} p^i z_i \geq b \tag{11.6}$$

$$\begin{aligned} \sum_{i \in \Lambda} z_i &= 1 \\ z &\geq 0 \end{aligned} \tag{11.7}$$

where  $\Lambda$  is a *very small* subset of the columns in the full master problem. We should have at least  $m_1 + 1$  columns in  $\Lambda$  in order to find a basis. Assume that the restricted master is feasible.

Let  $(u, u_0)$  be a set of optimal dual multipliers associated with the constraints  $\sum_{i=1}^q p^i z_i \geq b$  and  $\sum_{i=1}^q z_i = 1$ , respectively, in the restricted master. The reduced cost of variable  $z_i$  is  $f_i - u^\top p^i - u_0$ . If  $f_i - u^\top p^i - u_0 \geq 0$  for all  $i = 1, \dots, q$ , the optimal solution to the restricted master is the optimal solution to the full master. If at least one column prices out negative for  $i \notin \Lambda$ , then this column is added to the restricted master and the column that leaves the basis can be deleted from the restricted master. This eliminates the necessity of solving a linear program over the full master. Calculating the reduced cost explicitly for all the columns in the full master is not practical. But,

$$\begin{aligned} \min\{f_i - u^\top p^i - u_0 \mid i = 1, \dots, r\} &= \min\{c^\top x^i - u^\top A x^i - u_0 \mid i = 1, \dots, r\} \\ &= \min\{(c - A^\top u)^\top x^i \mid i = 1, \dots, r\} - u_0. \end{aligned}$$

Since there is always an optimal extreme point solution to a linear program when an optimum exists, finding the minimum reduced cost is equivalent to solving the following linear program

$$\begin{array}{ll} \min & (c - A^\top u)^\top x - u_0 \\ (DWS(u, u_0)) & \text{s.t.} \quad Bx \geq d \\ & \quad x \geq 0 \end{array}$$

Problem  $(DWS(u, u_0))$  is the Dantzig-Wolfe subproblem. After optimizing the restricted master, all the columns in the full master are priced out by solving the subproblem  $(DWS(u, u_0))$ . By assumption, the subproblem polyhedron  $\{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0\}$  is bounded, i.e. a polytope, so  $(DWS(u, u_0))$  cannot be unbounded. If  $(DWS(u, u_0))$  is infeasible, then the original linear program ( $LP$ ) is infeasible. Any such scheme of pricing where all columns are not explicitly priced out is called *column generation*. The linear program used to generate the columns is called the *subproblem*. This is why all of the rows in the original problem are not included in the matrix  $B$ . If this were the case, then the pricing problem would be equivalent to the original linear program. The objective is to put as many rows as possible into the  $B$  matrix, while leaving the constraint set  $Bx \geq d$  with as much special structure as possible in order to admit a fast solution. An illustration of this partitioning is given in the next section.

### Algorithm 11.3 (Dantzig-Wolfe Algorithm)

**Step 1:** Find an initial set of columns  $\Lambda$  and form the Dantzig-Wolfe restricted master program (DWR). Assume the restricted master is feasible.

**Step 2:** Optimize the restricted master program (DWR) and obtain an optimal set of dual multipliers  $\bar{u}, \bar{u}_0$ .

**Step 3:** Solve the Dantzig-Wolfe subproblem ( $DWS(u, u_0)$ ). Let  $x^j$  be an optimal basic solution with objective function value

$$(c - A^\top u)^\top x^j - u_0 = f_j - u^\top p^j - u_0.$$

If  $f_j - u^\top p^j - u_0 = 0$  stop, the basic solution corresponding to the current restricted master is optimal, else go to Step 4.

**Step 4:** Add the column  $\begin{bmatrix} p^j \\ 1 \end{bmatrix} = \begin{bmatrix} Ax^j \\ 1 \end{bmatrix}$  to the restricted master, update  $\Lambda \leftarrow \Lambda \cup \{j\}$  and go to Step 2.

There are numerous variations on this basic algorithm which affect the efficiency of the algorithm. For example, the column that  $j$  replaces in the basis can be deleted from the restricted master so that the restricted master does not grow in size. These issues are discussed at greater length in Section 11.5.

### 11.3 A LOCATION APPLICATION

Consider the following *simple plant location* model. This model was first introduced in Subsection 1.3.5 in Chapter 1 as the lock box location problem. See also, Section 10.3 in Chapter 10.

$$\min 2x_{11} + x_{12} + x_{13} + x_{21} + 2x_{22} + x_{23} + x_{31} + x_{32} + 2x_{33} + y_1 + y_2 + y_3$$

$$\text{s.t. } \begin{aligned} x_{11} + x_{21} + x_{31} &= 1 \\ x_{12} + x_{22} + x_{32} &= 1 \\ x_{13} + x_{23} + x_{33} &= 1 \end{aligned} \quad \left. \right\} Ax \geq b \text{ constraints}$$

$$\begin{aligned}
 & x_{11} \leq y_1 \leq 1 \\
 & x_{12} \leq y_1 \leq 1 \\
 & x_{13} \leq y_1 \leq 1 \\
 & x_{21} \leq y_2 \leq 1 \\
 & x_{22} \leq y_2 \leq 1 \\
 & x_{23} \leq y_2 \leq 1 \\
 & x_{31} \leq y_3 \leq 1 \\
 & x_{32} \leq y_3 \leq 1 \\
 & x_{33} \leq y_3 \leq 1
 \end{aligned} \quad \left. \right\} Bx \geq b \text{ constraints}$$

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

In this formulation,  $x_{ij} = 1$  if all of customer  $j$  demand is satisfied from plant  $i$ . The  $Bx \geq d$  constraints are setup forcing constraints. The forcing constraints require a plant to be open if there are shipments from that plant. See Subsection 1.3.5 in Chapter 1. In this example, we are interested in linear programming only and ignore the integer requirements on the  $x_{ij}$  and  $y_i$ .

If there are, for example, 25 potential plant sites and 300 customers then the  $A$  matrix would contain 300 rows but the  $B$  matrix would contain 7500 rows! Apply Algorithm 11.3 to this problem. A feasible restricted master is required to initialize the algorithm. Generate several extreme points of the subproblem.

Solution 1:  $y_1 = 1, x_{11} = x_{12} = x_{13} = 1$ , all others = 0.

Solution 2:  $y_1 = y_2 = y_3 = 1, x_{11} = x_{21} = x_{31} = 1$ , all others = 0.

Solution 3:  $y_1 = y_2 = y_3 = 1, x_{11} = x_{12} = x_{21} = x_{22} = x_{31} = x_{32} = 1$ , all others = 0.

Solution 4:  $y_1 = y_2 = y_3 = 1, x_{11} = x_{12} = x_{13} = x_{21} = x_{23} = x_{31} = x_{32} = 1$ , all others = 0.

Solution 5:  $y_2 = y_3 = 1, x_{21} = x_{31} = 1$ , all others = 0.

Taking the variables in the order given in the objective function results in  $A$  defined by

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

The last three columns of  $A$  are all zero since the  $y_i$  variables do not appear in the customer demand constraints which define  $A$ . Now take  $A$  and the five

subproblem solutions (which need not be extreme points) and generate the columns

$$p^i = A \begin{bmatrix} x \\ y \end{bmatrix}$$

$$p^1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} x_{11}^1 \\ x_{12}^1 \\ x_{13}^1 \\ x_{21}^1 \\ x_{22}^1 \\ x_{23}^1 \\ x_{31}^1 \\ x_{32}^1 \\ x_{33}^1 \\ y_1^1 \\ y_2^1 \\ y_3^1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

The other four columns are

$$p^2 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}, \quad p^3 = \begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix}, \quad p^4 = \begin{bmatrix} 3 \\ 2 \\ 2 \end{bmatrix}, \quad p^5 = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

The first element in each column is the coefficient for customer one and the last element is the coefficient for the customer three. This procedure is then repeated for the objective function coefficients which are

$$f_i = c^T \begin{bmatrix} x \\ y \end{bmatrix}$$

For example,

$$f_1 = [2 \ 1 \ 1 \ 1 \ 2 \ 1 \ 1 \ 1 \ 2 \ 1 \ 1 \ 1] = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} x_{11}^1 \\ x_{12}^1 \\ x_{13}^1 \\ x_{21}^1 \\ x_{22}^1 \\ x_{23}^1 \\ x_{31}^1 \\ x_{32}^1 \\ x_{33}^1 \\ y_1^1 \\ y_2^1 \\ y_3^1 \end{bmatrix} = 5$$

Similarly,  $f_2 = 7$ ,  $f_3 = 11$ ,  $f_4 = 11$  and  $f_5 = 4$ . The first restricted master is

$$\begin{array}{ll} \min & 5Z_1 + 7Z_2 + 11Z_3 + 11Z_4 + 4Z_5 \\ \text{s.t.} & \left. \begin{array}{lcl} Z_1 + 3Z_2 + 3Z_3 + 3Z_4 + 2Z_5 & = 1 \\ Z_1 + 3Z_3 + 2Z_4 & = 1 \\ Z_1 + 2Z_4 & = 1 \end{array} \right\} \text{coupling constraints} \\ & Z_1 + Z_2 + Z_3 + Z_4 + Z_5 = 1 \quad \text{convexity row} \\ & Z_1, Z_2, Z_3, Z_4, Z_5 \geq 0 \end{array}$$

#### OBJECTIVE FUNCTION VALUE

1) 5.0000000

VARIABLE	VALUE	REDUCED COST
Z1	1.000000	0.000000
Z2	0.000000	1.333333
Z3	0.000000	0.000000
Z4	0.000000	0.000000
Z5	0.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	-1.666667
3)	0.000000	-1.777778
4)	0.000000	-0.888889
5)	0.000000	-0.666667

The dual variables (to two decimal places) are  $u^T = [1.67, 1.78, 0.88]$ ,  $u_0 = .67$ . (In a minimization problem negate the dual prices to get the appropriate dual variables.) This yields in the following subproblem.

$$\begin{array}{l} \min .67x_{11} - .78x_{12} + .12x_{13} - .67x_{21} + .23x_{22} + .12x_{23} - .67x_{31} - .78x_{32} + \\ 1.12x_{33} + y_1 + y_2 + y_3 - .67 \end{array}$$

$$\begin{array}{l} \text{s.t. } 0 \leq x_{11} \leq y_1 \leq 1, \quad 0 \leq x_{21} \leq y_2 \leq 1, \quad 0 \leq x_{31} \leq y_3 \leq 1 \\ 0 \leq x_{12} \leq y_1 \leq 1, \quad 0 \leq x_{22} \leq y_2 \leq 1, \quad 0 \leq x_{32} \leq y_3 \leq 1 \\ 0 \leq x_{13} \leq y_1 \leq 1, \quad 0 \leq x_{23} \leq y_2 \leq 1, \quad 0 \leq x_{33} \leq y_3 \leq 1 \end{array}$$

Notice that the objective function coefficients of the variables are different than in the original statement of the problem. The objective function of the

subproblem is  $(c - A^T u)^T x - u_0$ . For example, consider the coefficient of  $1/3$  on variable  $x_{11}$ . This coefficient is the first component of  $(c - A^T u)$  which is

$$2 - [1, 0, 0] \begin{bmatrix} 1.67 \\ 1.78 \\ 0.89 \end{bmatrix}.$$

The optimal solution<sup>1</sup> to the subproblem is  $x_{31} = x_{32} = 1$ ,  $y_3 = 1$ , all others = 0. The optimal solution value is -1.45. Since the column corresponding to this solution prices out negative, it is added to the restricted master. This column is

$$p^6 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

with  $f_6 = 3$ . The new master and corresponding LP solution is

```

MIN      5 Z1 + 7 Z2 + 11 Z3 + 11 Z4 + 4 Z5 + 3 Z6
SUBJECT TO
    2) Z1 + 3 Z2 + 3 Z3 + 3 Z4 + 2 Z5 + Z6 =     1
    3) Z1 + 3 Z3 + 2 Z4 + Z6 =     1
    4) Z1 + 2 Z4 =     1
    5) Z1 + Z2 + Z3 + Z4 + Z5 + Z6 =     1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 5.00000000

VARIABLE	VALUE	REDUCED COST
Z1	1.000000	0.000000
Z2	0.000000	1.333333
Z3	0.000000	3.333333
Z4	0.000000	0.000000
Z5	0.000000	0.000000
Z6	0.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	-1.666667
3)	0.000000	-0.666667
4)	0.000000	-2.000000

---

<sup>1</sup>It is not necessary to use linear programming to solve the subproblem.

5) 0.000000 -0.666667

The solution value has not changed. Why? The new dual variables are  $u^T = [1.67, .67, 2]$ ,  $u_0 = .67$ . This gives the following subproblem

$$\min .33x_{11} + .33x_{12} - x_{13} - .67x_{21} + 1.33x_{22} - x_{23} - .67x_{31} + .33x_{32} + 0x_{33} + y_1 + y_2 + y_3 - .67$$

$$\text{s.t. } \begin{aligned} 0 \leq x_{11} &\leq y_1 \leq 1, & 0 \leq x_{21} &\leq y_2 \leq 1, & 0 \leq x_{31} &\leq y_3 \leq 1 \\ 0 \leq x_{12} &\leq y_1 \leq 1, & 0 \leq x_{22} &\leq y_2 \leq 1, & 0 \leq x_{32} &\leq y_3 \leq 1 \\ 0 \leq x_{13} &\leq y_1 \leq 1, & 0 \leq x_{23} &\leq y_2 \leq 1, & 0 \leq x_{33} &\leq y_3 \leq 1 \end{aligned}$$

An optimal solution is  $x_{21} = 1$ ,  $x_{23} = 1$ ,  $y_2 = 1$ , all others = 0. The solution value is  $-1.33$  so add the corresponding column. The new column is

$$p^7 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

with  $f_7 = 3$  The new master and corresponding solution is:

```

MIN      5 Z1 + 7 Z2 + 11 Z3 + 11 Z4 + 4 Z5 + 3 Z6 + 3 Z7
SUBJECT TO
    2) Z1 + 3 Z2 + 3 Z3 + 3 Z4 + 2 Z5 + Z6 + Z7 =     1
    3) Z1 + 3 Z3 + 2 Z4 + Z6 =     1
    4) Z1 + 2 Z4 + Z7 =     1
    5) Z1 + Z2 + Z3 + Z4 + Z5 + Z6 + Z7 =     1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 5.00000000

VARIABLE	VALUE	REDUCED COST
Z1	1.000000	0.000000
Z2	0.000000	4.000000
Z3	0.000000	2.000000
Z4	0.000000	0.000000
Z5	0.000000	2.000000

Z6	0.000000	0.000000
Z7	0.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	-1.000000
3)	0.000000	-2.000000
4)	0.000000	-2.000000
5)	0.000000	0.000000

The new dual variables are  $u^T = [1, 2, 2]$ ,  $u_0 = 0$ . The optimal solution to the corresponding subproblem  $y_1 = 1$ ,  $x_{12} = x_{13} = 1$  for a reduced cost of -1. Add the corresponding column

$$p^8 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

with  $f_8 = 3$  and solve.

```

MIN      5 Z1 + 7 Z2 + 11 Z3 + 11 Z4 + 4 Z5 + 3 Z6 + 3 Z7
      + 3 Z8
SUBJECT TO
    2) Z1 + 3 Z2 + 3 Z3 + 3 Z4 + 2 Z5 + Z6 + Z7 =   1
    3) Z1 + 3 Z3 + 2 Z4 + Z6 + Z8 =     1
    4) Z1 + 2 Z4 + Z7 + Z8 =     1
    5) Z1 + Z2 + Z3 + Z4 + Z5 + Z6 + Z7 + Z8 =   1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 4.59999990

VARIABLE	VALUE	REDUCED COST
Z1	0.000000	0.400000
Z2	0.000000	2.400000
Z3	0.000000	1.600000
Z4	0.200000	0.000000
Z5	0.000000	1.000000
Z6	0.200000	0.000000
Z7	0.200000	0.000000
Z8	0.400000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	-1.600000
3)	0.000000	-1.600000
4)	0.000000	-1.600000
5)	0.000000	0.200000

The new dual variables are  $u = [1.6, 1.6, 1.6]^T$ ,  $u_0 = 0.2$ . The optimal solution to the subproblem is  $x_{12} = x_{13} = x_{21} = x_{23} = x_{32} = x_{31} = 1$ ,  $y_1 = y_2 = y_3 = 1$ . The solution value is -4. Add the corresponding column

$$p^9 = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

with  $f_9 = 9$ . The new master program and solution are

```

MIN      5 Z1 + 7 Z2 + 11 Z3 + 11 Z4 + 4 Z5 + 3 Z6 + 3 Z7
      + 3 Z8 + 9 Z9
SUBJECT TO
      2)   Z1 + 3 Z2 + 3 Z3 + 3 Z4 + 2 Z5 + Z6 + Z7 + 2 Z9
      =     1
      3)   Z1 + 3 Z3 + 2 Z4 + Z6 + Z8 + 2 Z9 =     1
      4)   Z1 + 2 Z4 + Z7 + Z8 + 2 Z9 =     1
      5)   Z1 + Z2 + Z3 + Z4 + Z5 + Z6 + Z7 + Z8 + Z9 =     1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 4.5000000

VARIABLE	VALUE	REDUCED COST
Z1	0.000000	0.500000
Z2	0.000000	2.500000
Z3	0.000000	2.000000
Z4	0.000000	0.500000
Z5	0.000000	1.000000
Z6	0.250000	0.000000
Z7	0.250000	0.000000
Z8	0.250000	0.000000
Z9	0.250000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	-1.500000
3)	0.000000	-1.500000
4)	0.000000	-1.500000
5)	0.000000	0.000000

The new subproblem is

$$\min .5x_{11} - .5x_{12} - .5x_{13} - .5x_{21} + .5x_{22} - .5x_{23} - .5x_{31} - .5x_{32} + .5x_{33} + y_1 + y_2 + y_3$$

$$\text{s.t. } \begin{aligned} 0 \leq x_{11} &\leq y_1 \leq 1, & 0 \leq x_{21} &\leq y_2 \leq 1, & 0 \leq x_{31} &\leq y_3 \leq 1 \\ 0 \leq x_{12} &\leq y_1 \leq 1, & 0 \leq x_{22} &\leq y_2 \leq 1, & 0 \leq x_{32} &\leq y_3 \leq 1 \\ 0 \leq x_{13} &\leq y_1 \leq 1, & 0 \leq x_{23} &\leq y_2 \leq 1, & 0 \leq x_{33} &\leq y_3 \leq 1 \end{aligned}$$

The optimal solution value is zero. Therefore, the optimal solution to the restricted master is the optimal solution to the full master. The optimal solution is  $Z_6 = Z_7 = Z_8 = Z_9 = 0.25$ . The corresponding solution to our original problem is

$$(25) \quad \begin{array}{c|c|c|c|c} \left[ \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{matrix} \right] & + (.25) \left[ \begin{matrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \right] & + (.25) \left[ \begin{matrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{matrix} \right] & + (.25) \left[ \begin{matrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{matrix} \right] & = \left[ \begin{matrix} 0 \\ 1/2 \\ 1/2 \\ 1/2 \\ 0 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \\ 0 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{matrix} \right] \end{array}$$

The optimal solution in the  $(x, y)$  variables is

$$\begin{aligned} y_1 &= y_2 = y_3 = \frac{1}{2} \\ x_{12} &= x_{13} = x_{21} = x_{23} = x_{31} = x_{32} = \frac{1}{2} \\ x_{11} &= x_{22} = x_{33} = 0. \end{aligned}$$

Notice that this optimal solution is a convex combination of extreme points of the feasible region defined by  $Ax \geq b$ ,  $x \geq 0$ . This is the case whenever two or more of the  $z_i$  master variables are strictly positive.

## 11.4 TAKING ADVANTAGE OF BLOCK ANGULAR STRUCTURE

In the current development the constraint matrix is partitioned into two parts,  $A$  and  $B$ . This partitioning is inconsequential to the theoretical development. However, selecting the pivot column requires solving a linear program with constraint set  $Bx \geq d$  so for an efficient implementation,  $B$  should have as much special structure as possible. Block angular linear programs provide a natural partition of the original constraint set into an  $A$  and  $B$ . A *block angular linear program* has the following structure.

$$(BAP) \quad \begin{array}{lll} \min & c^1 x^1 + c^2 x^2 + \cdots + c^m x^m \\ \text{s.t.} & A_1 x^1 + A_2 x^2 + \cdots + A_m x^m \geq b \\ & B_1 x^1 \geq d^1 \\ & B_2 x^2 \geq d^2 \\ & \vdots \\ & B_m x^m \geq d^m \\ x^1, x^2, \dots, x^m & \geq 0 \end{array}$$

Assume that  $(BAP)$  is difficult to solve by the direct application of either a barrier algorithm or the revised simplex algorithm. This might be true if  $m$  and/or the  $D^i$  are large. Partition the constraint matrix of  $(BAP)$  into  $A$  and  $B$  as follows.

$$A = [ A_1 \ A_2 \ \cdots \ A_m ]$$

and

$$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix}.$$

The  $B$  matrix has very special structure. The subproblem can be optimized by solving  $m$  *independent* linear programs. Applying a barrier algorithm or the revised simplex algorithm  $m$  times, once to each block, is much more efficient than applying it to the entire  $Bx \geq d$  system. Why is this so? There is another interesting aspect to the block angular structure. It is possible to create a

master problem exactly as before where each  $p^j$  is

$$p^j = \begin{bmatrix} B_1 & B_2 & \cdots & B_m \end{bmatrix} \begin{bmatrix} x^{1j} \\ x^{2j} \\ \vdots \\ x^{mj} \end{bmatrix},$$

where  $x^{ij}$  is the  $j$ th extreme point of the polytope defined by the equations

$$B_i x^i \geq d^i, \quad x^i \geq 0.$$

Assuming these polyhedra are bounded, by the finite basis theorem any solution to  $B_i x^i \geq d^i, x^i \geq 0$  is expressed as

$$x^i = \sum_{j=1}^{q_i} z_{ij} x^{ij}, \quad \sum_{j=1}^{q_i} z_{ij} = 1, \quad z_{ij} \geq 0.$$

Using the same logic as in Section 11.2, and substituting  $\sum_{j=1}^{q_i} z_{ij} x^{ij}$  for  $x^i$ , in (BAP) gives the following master program.

$$\begin{aligned} \min \quad & \sum_{i=1}^m \sum_{j=1}^{q_i} f_{ij} z_{ij} \\ \text{s.t.} \quad & \sum_{i=1}^m \sum_{j=1}^{q_i} p^{ij} z_{ij} \geq b \\ & \sum_{j=1}^{q_i} z_{ij} = 1, \quad i = 1, \dots, m \\ & z_{ij} \geq 0, \quad j = 1, \dots, q_i, \quad i = 1, \dots, m \end{aligned}$$

where  $f_{ij} = (c^i)^\top x^{ij}$  and  $p^{ij} = B^i x^{ij}$ . This master program has the same number of coupling constraints as the master program given by (11.1)-(11.4), only now there are  $m$  convexity, or GUB, or SOS rows instead of just one. There are also  $m$  subproblems to solve instead of just one. Letting  $u_{i0}$  denote the dual variable associated with the  $i$ th convexity row, the  $i$ th subproblem is

$$\begin{aligned} \min \quad & (c^i - (A_i)^\top u)^\top x^i - u_{i0} \\ & B_i x^i \geq d^i, \quad i = 1, \dots, m \\ & x^i \geq 0, \quad i = 1, \dots, m. \end{aligned}$$

In the homework exercises you are asked to compare the computational advantages and disadvantages of an  $m$  convexity master versus a one convexity master.

## 11.5 COMPUTATIONAL ISSUES

Several of the computational issues related to actually implementing a Dantzig-Wolfe decomposition algorithm are discussed in this section.

### 11.5.1 Pivoting and Column Selection Strategies

*Master Problem Size:* If  $A$  has  $m'$  rows and there are  $m$  convexity rows, it is necessary to keep only  $m' + m$  columns in the master in order to have a basis. However, it is generally best to keep more columns in the restricted master than necessary. This allows the possibility of performing the pricing operation over only the restricted master columns without the necessity of solving any subproblems. It is only necessary to solve the subproblems when all of the columns in the restricted master price out to a nonnegative value. Of course, the user should put some limit on the growth of the restricted master and purge old columns that have been inactive for many iterations. This strategy is similar to the partial pricing strategy described in Subsection 6.6.1 of Chapter 6.

*Column generation for  $m$  convexity row master programs:* There are (at least) two strategies here. One possibility is to solve each subproblem and then add to the restricted master a column for each subproblem that prices out negatively. An alternative to this, is to solve each subproblem, but add only the column with the largest negative reduced cost in absolute value. When there are  $m$  subproblems there are between one and  $m$  potential columns to add at each iteration.

*Bringing in near optimal columns:* It is possible to bring in more than one column for each subproblem. Any extreme point solution of a subproblem which gives a negative reduced cost has the potential to improve the optimal value of the restricted master. Thus, a set of near optimal columns for each subproblem could be added to the restricted master at each iteration of the problem. Ho and Louie [231] discuss an implementation of Dantzig-Wolfe decomposition where they provide four user parameters,  $q_{freq}$ ,  $q_{perc}$ ,  $q_{max}$ , and  $q_{term}$ . The proposal generation procedure is triggered by the first feasible solution  $x^{ij}$  to subproblem  $i$  which satisfies

$$\gamma_i = (c^i - (A^i)^\top u)^\top x^{ij} - u_{i0} < 0.$$

The corresponding column is generated. Then a column is generated every  $q_{freq}$  iterations, or whenever  $\gamma_i$  is decreased by  $q_{perc}$ . No more than  $q_{max}$  proposals

are kept for transmission. The procedure is terminated once  $q_{term}$  proposals have been generated. The last  $q_{max}$  proposals are then sent to the master problem.

*Solving the subproblem to optimality:* It is not necessary to solve the subproblems to optimality. It is only necessary to add a column corresponding to a negative reduced cost. Notice that Ho and Loute terminate solution of the subproblem once  $q_{term}$  proposals have been generated. There is no guarantee that the optimal solution to the subproblem has been found at this point.

*Software Availability:* At this point in time, IBM's OSL is the only commercial software we are aware of that implements Dantzig-Wolfe decomposition. There are several reasons for this.

1. Software developers cater to the average user, and the average user seldom has problems large enough to warrant decomposition. On the other hand, the user is unlikely to formulate problems too large for available software. Your basic Catch-22.
2. Users with truly large problems (e.g. lot sizing, routing, cutting stock) develop specialized decomposition techniques that are not easily generalized. Since the efficiency of Dantzig-Wolfe depends on solving the subproblems fast it is necessary to develop specialized algorithms for solving the subproblems. These special algorithms may not generalize easily to a wide variety of applications.
3. Recent advances with both simplex and barrier codes make it possible to solve very large problems without resorting to decomposition methods.

The OSL package has several nice features. If the subproblems have a network structure, then a network algorithm is used on these. See Section 14.3 in Chapter 14. The subproblems are also used in a crashing procedure in an attempt to find a good starting solution. See Section 6.5 in Chapter 6. For more information on the OSL procedures see the following Web site.

<http://www6.software.ibm.com/es/oslv2/features/featur09.htm#HDRLPDPC>

### 11.5.2 Establishing Bounds Short of Optimality

When solving very large linear programs column generation algorithms may experience diminishing returns. That is, the objective function value as a function of the number of pivots has a negative first derivative (assume minimization objective function), but a positive second derivative. Therefore, it is often desirable to terminate the solution procedure if we are within some tolerance of the optimal solution. It is easy to establish upper and lower bounds on the optimal objective function value when using Dantzig-Wolfe decomposition.

**Upper Bound:** Any feasible solution to a minimization problem provides an upper bound on the optimal solution value. Therefore, at each iteration the current solution value to the restricted master provides an upper bound on the optimal solution value since it is a restriction of the full master. The optimal solution value to the restricted master is monotonic non-increasing at each iteration.

**Lower Bound:** Let  $z_0(LPR(u))$  denote the optimal solution value to the linear program  $(LPR(u))$  defined by

$$(LPR(u)) \quad \begin{aligned} \min \quad & (c - A^T u)^T x + b^T u \\ \text{subject to} \quad & Bx \geq d \\ & x \geq 0 \end{aligned}$$

and let  $z_0(LP)$  denote the optimal solution value to the original linear program  $(LP)$  defined at the start of Section 11.1.

**Lemma 11.4** *For nonnegative  $u$ , linear program  $(LPR(u))$  is a relaxation of linear program  $(LP)$  and  $z_0(LPR(u)) \leq z_0(LP)$ .*

**Proof:** If  $(LP)$  is infeasible the result is vacuously true. Assume  $(LP)$  is feasible. Let  $\bar{x}$  be any feasible solution to  $(LP)$ . Then  $A\bar{x} \geq b$  which implies for nonnegative  $u$ ,

$$c^T \bar{x} \geq c^T \bar{x} + u^T (b - A\bar{x}) = (c - A^T u)^T \bar{x} + b^T u.$$

Furthermore,  $\bar{x}$  feasible to  $(LP)$  implies  $\bar{x}$  is a feasible solution to  $(LPR(u))$ . Then any feasible solution to  $(LP)$  is feasible to  $(LPR(u))$  and their objective function values are equal.  $\square$

Now let  $z_0(SP(u, u_0))$  denote the optimal value of the Dantzig-Wolfe subproblem defined by  $(DWS)$  in Section 11.2. Observe

$$z_0(LPR(u)) = z_0(DWS(u, u_0)) + u_0 + b^\top u.$$

This observation combined with Lemma 11.4 on the lower bound gives the main result.

**Proposition 11.5** *If  $\bar{x}$  is a feasible solution to the restricted Dantzig-Wolfe master then for any nonnegative  $u$ ,*

$$c^\top \bar{x} \geq z_0(LP) \geq z_0(DWS(u, u_0)) + u_0 + b^\top u. \quad (11.8)$$

Thus, at every step of the Dantzig-Wolfe algorithm the value of the restricted master provides an upper bound on the optimal solution value of  $(LP)$  and if  $u, u_0$  are the optimal dual variables on the coupling and convexity constraints, respectively, adding  $u_0 + b^\top u$  to the optimal value of the subproblem,  $z_0(DWS(u, u_0))$ , provides a lower bound on the optimal solution value of  $(LP)$ . When the difference of these bounds is within an acceptable  $\epsilon$  the algorithm is terminated.

### 11.5.3 Obtaining a Starting Solution

It is necessary to initialize the restricted master with a set of columns such that the restricted master is feasible. In many applications, such as the simple plant location problem, it is easy to generate a set of initial columns such that the restricted master is feasible. When this is not the case, a Phase I procedure is required. See Section 6.5 in Chapter 6.

### 11.5.4 How to Handle an Unbounded Subproblem

In the development so far it was assumed that the constraints  $Bx \geq d, x \geq 0$  defined a polytope. This is important when solving the subproblem  $(DWS(u, u_0))$  since the polytope assumption guarantees that the subproblem is never unbounded. However, even in the more general case of a polyhedron, the problem of an unbounded solution is resolved by using the finite basis theorem. In general, the polyhedron defined by  $Bx \geq d, x \geq 0$  can be written as a convex

plus conic combination of extreme points and rays, respectively. Let  $x^1, \dots, x^q$  denote the extreme points of the polyhedron and  $x^{q+1}, \dots, x^r$  the extreme rays of the recession cone of the polyhedron. Then if  $\bar{x}$  is a point in the polyhedron

$$\bar{x} = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i, \quad \sum_{i=1}^q z_i = 1, \quad z_i \geq 0, \quad i = 1, \dots, r.$$

Using the same logic as before, the Dantzig-Wolfe master equivalent to problem (LP) is

$$\begin{aligned} \min \quad & \sum_{i=1}^r f_i z_i \\ \text{s.t.} \quad & \sum_{i=1}^r p^i z_i \geq b \\ & \sum_{i=1}^q z_i = 1 \\ & z_i \geq 0, \quad i = 1, \dots, r \end{aligned}$$

There still remains the problem of exactly what to do in terms of column generation when a set of dual variables  $(u, u_0)$  are generated which lead to an unbounded subproblem. This is left as Exercise 11.7.

## 11.6 CONCLUSION

Dantzig and Wolfe [112] proposed their decomposition method as an algorithm for solving very large linear programs. Recent advances in numerical linear algebra (see Chapter 13) incorporated within simplex and barrier algorithms have certainly pushed back the boundary on the size of linear programs which can be solved without resorting to decomposition methods. However, the column generation philosophy of using a subproblem to generate columns on an as needed basis is still very important and used frequently in the solution of large mathematical programs. This strategy is very effective when the subproblems have special structure and are easily optimized. See Chapter 15 for examples of column generation used to solve large integer linear programs.

This chapter was written from a simplex point of view. See Goffin and Vial [187], Todd [425], and Ye [468] for column generation within an interior point framework.

## 11.7 EXERCISES

- 11.1 Develop an algorithm for solving the subproblem used in the simple plant location model of Section 11.3. This algorithm should not require the use of simplex or interior point methods.
- 11.2 In the presence of primal degeneracy, it is possible that adding a column from the subproblem with a negative reduced cost does not result in a change in the objective function value or the primal solution. Is it possible that there is also no change in the dual solution? If you answer yes, show how this could happen. If you answer no, provide a proof for why this cannot happen.
- 11.3 Is possible to generate *the same* column at consecutive iterations of the Dantzig-Wolfe algorithm? If a column is deleted from the restricted master is it possible to generate this column again at a later iteration?
- 11.4 Take the linear program

$$\begin{array}{lll} \max & 10x_1 + 9x_2 \\ \text{s.t.} & (7/10)x_1 + x_2 & \leq 630 \\ & (1/2)x_1 + (5/6)x_2 & \leq 600 \\ & x_1 + (2/3)x_2 & \leq 708 \\ & (1/10)x_1 + (1/4)x_2 & \leq 135 \\ & x_1, x_2 & \geq 0 \end{array}$$

from Anderson, Sweeney and Williams [12] and create the full Dantzig-Wolfe master. Treat the first two constraints as the coupling constraints and the third and fourth constraints (along with nonnegativity) as the subproblem constraints. Solve the resulting linear program.

- 11.5 Apply the Dantzig-Wolfe decomposition algorithm to the simple plant location problem data in Section 11.3. Solve the problem using a Dantzig-Wolfe master where there is one convexity row for every block (there is one block for each plant). That is, use the master program for the block angular problem (*BAP*) in Section 11.4.
- 11.6 Assume you have a block angular problem with  $m$  blocks. One strategy is to form a master with only one convexity constraint, another is to form a master with  $m$  convexity constraints. Discuss the relative computational advantages and disadvantages of each approach.
- 11.7 If the subproblem feasible region is a polyhedron, then it may have an unbounded solution for a given  $(u, u_0)$ . Describe how you would identify

this situation using the simplex algorithm. Then *very clearly* describe how to use the subproblem tableau to generate a column to add to the master which corresponds to an extreme ray solution.

## **PART V**

---

### **SOLVING LARGE SCALE PROBLEMS: USING SPECIAL STRUCTURE**

---

## LAGRANGIAN METHODS

### 12.1 INTRODUCTION

In Chapter 9 we introduced the concept of a problem *relaxation*. Problem

$$(PR) \quad \min\{f(x) \mid x \in \Gamma \subseteq \mathbb{R}^n\}$$

is a relaxation of problem

$$(P) \quad \min\{g(x) \mid x \in \hat{\Gamma} \subseteq \mathbb{R}^n\}$$

if and only if  $\hat{\Gamma} \subseteq \Gamma$  and  $f(x) \leq g(x)$  for all  $x \in \hat{\Gamma}$ . It follows that if  $(PR)$  is a relaxation of  $(P)$ , the optimal solution value of  $(PR)$  is less than or equal to the optimal solution value of  $(P)$ . If  $(PR)$  is relaxation of  $(P)$ , then  $(P)$  is a *restriction* of  $(PR)$ . In Chapter 9 a branch-and-bound algorithm was presented for solving integer linear programs based upon the linear programming relaxation of an integer program. Linear programming is a natural relaxation to use because of the availability of very fast routines for solving linear programs. Unfortunately, the linear programming relaxation may be a very poor approximation of the original problem making enumeration an untenable solution approach. In this chapter we develop a new type of relaxation based upon the *Lagrangian dual*. Approaches for improving the linear programming relaxation value are covered in Chapters 15 and 16.

In Section 12.2 we give the formal definition of the Lagrangian dual and show its relation to the linear programming dual. In Section 12.3 we apply the ideas developed in Section 12.2 to integer linear programming. In Section 12.4 we investigate important properties of the Lagrangian dual and prove that it is a concave function. This concavity property is used in Section 12.5 for developing

algorithms to optimize the Lagrangian dual. The key computational issues associated with using the Lagrangian dual are discussed in Section 12.6. In Section 12.7 the Lagrangian dual is used to develop an analog of the Dantzig-Wolfe decomposition algorithm for integer programming. Concluding remarks are given in Section 12.8. Exercises are provided in Section 12.9.

In this chapter,  $z_0(*)$  denotes the optimal solution value of problem  $(*)$  and  $(\bar{*})$  is the linear programming relaxation of problem  $(*)$ .

## 12.2 THE LAGRANGIAN DUAL

In Chapter 11 on Dantzig-Wolfe decomposition we applied inverse projection to the constraint set  $Bx \geq d$  in linear program

$$(LP) \quad \begin{aligned} & \min c^\top x \\ & \text{s.t. } Ax \geq b \\ & \quad Bx \geq d \\ & \quad x \geq 0 \end{aligned}$$

Applying inverse projection gives the equivalent Dantzig-Wolfe master linear program

$$(DW) \quad \begin{aligned} & \min \sum_{i=1}^r f_i z_i \\ & \text{s.t. } \sum_{i=1}^r p^i z_i \geq b \\ & \quad \sum_{i=1}^q z_i = 1 \\ & \quad z_i \geq 0, \quad i = 1, \dots, r \end{aligned}$$

where  $p^i = Ax^i$ ,  $i = 1, \dots, r$ ,  $f_i = c^\top x^i$ ,  $i = 1, \dots, r$  and  $x^i$ ,  $i = 1, \dots, q$  are the extreme points of the polyhedron  $P = \{x | Bx \geq d, x \geq 0\}$  and  $x^i$ ,  $i = q+1, \dots, r$  are the extreme rays of the recession cone of the polyhedron  $P$ . It is not necessary to include all of the variables in model  $(DW)$  and variables can be added “on the fly” as needed. In this chapter we study another approach for optimizing  $(DW)$ . This approach is based on the Lagrangian dual and extends directly to integer and nonlinear programming.

Let  $u$  be a vector of dual variables for constraint set  $\sum_{i=1}^r p^i z_i \geq b$  and  $u_0$  a dual variable for constraint  $\sum_{i=1}^q z_i = 1$ . The dual of the linear program (DW) is

$$(DDW) \quad \begin{aligned} \max \quad & u_0 + b^\top u \\ \text{s.t.} \quad & u_0 + u^\top p^i \leq f_i, \quad i = 1, \dots, q \\ & u^\top p^i \leq f_i, \quad i = q+1, \dots, r \\ & u \geq 0. \end{aligned}$$

In any optimal solution to (DDW),  $u_0$  will take on the minimum value of  $f_i - u^\top p^i$  over  $i = 1, \dots, q$ . Therefore, (DDW) is equivalent to the following problem.

$$\max \{ u^\top b + \min\{(f_i - u^\top p^i) \mid i = 1, \dots, q\} \mid u \geq 0, 0 \leq f_i - u^\top p^i, \\ i = q+1, \dots, r \}$$

Since the  $x^i, i = 1, \dots, r$  are the extreme points and rays of the polyhedron  $\{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0\}$  and  $f_i = c^\top x^i$ ,  $p^i = Ax^i$  it follows from Theorem 3.4, the fundamental theorem of linear programming, that

$$\max \{ u^\top b + \min\{(f_i - u^\top p^i) \mid i = 1, \dots, r\} \mid u \geq 0, \\ u^\top p^i \leq f_i, i = q+1, \dots, r \} \quad (12.1)$$

$$= \max \{ u^\top b + \min\{(c - A^\top u)^\top x \mid Bx \geq d, x \geq 0\} \mid u \geq 0 \} \quad (12.2)$$

The constraints  $u^\top p^i \leq f_i, i = q+1, \dots, r$  are deleted when going from (12.1) to (12.2) since  $(f_i - u^\top p^i) < 0$  for an  $i \geq q+1$  corresponds to an extreme ray  $x^i$  such that  $(c - A^\top u)^\top x^i < 0$ . Then for this value of  $u$  the inner minimization  $\min\{(c - A^\top u)^\top x \mid Bx \geq d, x \geq 0\}$  goes to negative infinity. Further, because the  $b^\top u$  term does not affect the inner minimization,

$$\begin{aligned} & \max_{u \geq 0} \{ b^\top u + \min\{(c - A^\top u)^\top x \mid Bx \geq d, x \geq 0\} \} \\ &= \max_{u \geq 0} \{ \min\{(c - A^\top u)^\top x + b^\top u \mid Bx \geq d, x \geq 0\} \}. \end{aligned}$$

The optimization problem  $\max_{u \geq 0} L(u)$  where

$$L(u) := \min\{(c - A^\top u)^\top x + b^\top u \mid Bx \geq d, x \geq 0\} \quad (12.3)$$

is the *Lagrangian dual* of the linear program

$$(LP) \quad \begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq b \\ & Bx \geq d \\ & x \geq 0. \end{aligned}$$

We say we have “dualized” the constraints  $Ax \geq b$  and formed a *Lagrangian function*  $L(u)$ . The results of this development are summarized in the following Proposition.

**Proposition 12.1** *If the linear program  $\min\{c^T x \mid Ax \geq b, Bx \geq d, x \geq 0\}$  has an optimal solution and  $\bar{u}$  is an optimal set of linear programming dual multipliers on the constraint set  $Ax \geq b$ , then*

$$\min\{c^T x \mid Ax \geq b, Bx \geq d, x \geq 0\} = \max\{L(u) \mid u \geq 0\} = L(\bar{u})$$

where  $L(u) = \min\{(c - A^T u)^T x + b^T u \mid Bx \geq d, x \geq 0\}$ .

**Proof:** Let  $z_0(LP)$  denote the optimal solution value of  $(LP)$  and  $z_0(DW)$  the optimal solution value of  $(DW)$ . By inverse projection,  $z_0(LP) = z_0(DW)$ . By strong duality,  $z_0(DW) = z_0(DDW)$ . From (12.1)-(12.2) and the definition of  $L(u)$ ,  $z_0(LP) = \max\{L(u) \mid u \geq 0\}$ .  $\square$

Before extending the Lagrangian dual to integer programming consider the following simple example.

**Example 12.2** *This example is a continuation of Example 11.2 from the Chapter 11. The example linear program is*

$$\begin{aligned} \min & -x_1 - x_2 \\ -x_1 + x_2 & \geq -2 \\ -4x_1 - 9x_2 & \geq -18 \\ 2x_1 - 4x_2 & \geq -4 \\ x_1, x_2 & \geq 0 \end{aligned}$$

Again, treat the first constraint as the constraint set  $Ax \geq b$  and the second and third constraints as the constraint set  $B \geq d$ . The theoretical development so far does not depend in any way on the constraints of the linear program having any special form, i.e. equality, less than or equal, or greater than or equal.

There are four extreme points in the polytope defined by the second and third constraints, along with the nonnegativity constraints. The coordinates of these four points are  $(0, 0)$ ,  $(4(1/2), 0)$ ,  $(1(1/17), 1(9/17))$  and  $(0, 1)$ . Any point in this polytope can be written as

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = z_1 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + z_2 \begin{bmatrix} 4(1/2) \\ 0 \end{bmatrix} + z_3 \begin{bmatrix} 1(1/17) \\ 1(9/17) \end{bmatrix} + z_4 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where  $z_1 + z_2 + z_3 + z_4 = 1$  and are nonnegative.

The columns in the master are  $p^i = Ax^i$ ,  $i = 1, 2, 3, 4$ . Since  $A = [-1, 1]$ , this gives

$$p^1 = [-1, 1] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = [ \ 0 \ ], p^2 = [-1, 1] \begin{bmatrix} 4(1/2) \\ 0 \end{bmatrix} = [ \ -4(1/2) \ ],$$

$$p^3 = [-1, 1] \begin{bmatrix} 1(1/17) \\ 1(9/17) \end{bmatrix} = [ \ (8/17) \ ], p^4 = [-1, 1] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [ \ 1 \ ].$$

The objective function coefficients in the master are  $c^\top x^i = [-1, -1]x^i$ ,  $i = 1, 2, 3, 4$ . This gives

$$f_1 = [-1, -1] \begin{bmatrix} 0 \\ 0 \end{bmatrix} = [ \ 0 \ ], f_2 = [-1, -1] \begin{bmatrix} 4(1/2) \\ 0 \end{bmatrix} = [ \ -4(1/2) \ ],$$

$$f_3 = [-1, -1] \begin{bmatrix} 1(1/17) \\ 1(9/17) \end{bmatrix} = [ \ -2(10/17) \ ], f_4 = [-1, -1] \begin{bmatrix} 0 \\ 1 \end{bmatrix} = [ \ -1 \ ].$$

The master program is

$$\begin{array}{llll} \min & 0z_1 - 4(1/2)z_2 - 2(10/17)z_3 - z_4 \\ \text{s.t.} & 0z_1 - 4(1/2)z_2 + (8/17)z_3 + z_4 & \geq & -2 \\ & z_1 + z_2 + z_3 + z_4 & = & 1 \\ & z_1, z_2, z_3, z_4 & \geq & 0 \end{array}$$

The dual of this master is

$$\begin{array}{lll} \max & -2u + u_0 \\ \text{s.t.} & u_0 & \leq 0 \end{array}$$

$$\begin{aligned} u_0 &\leq -4\frac{1}{2} + 4\frac{1}{2}u \\ u_0 &\leq -2\frac{10}{17} - \frac{8}{17}u \\ u_0 &\leq -1 - u \\ u &\geq 0 \end{aligned}$$

This is equivalent to

$$\max\{-2u + \min\{0, -4\frac{1}{2} + 4\frac{1}{2}u, -2\frac{10}{17} - \frac{8}{17}u, -1 - u\} \mid u \geq 0\}$$

which is equivalent to

$$\begin{aligned} \max \{ -2u + \min\{x_1(-1+u) - x_2(1+u) \mid -4x_1 - 9x_2 \geq -18, \\ 2x_1 - 4x_2 \geq -4, x_1, x_2 \geq 0\} \mid u \geq 0\} = \max\{L(u) \mid u \geq 0\} \end{aligned}$$

where corresponding Lagrangian function  $L(u)$  is

$$\begin{aligned} L(u) = -2u + \min\{x_1(-1+u) - x_2(1+u) \mid -4x_1 - 9x_2 \geq -18, \\ 2x_1 - 4x_2 \geq -4, x_1, x_2 \geq 0\}. \end{aligned}$$

The optimal dual solution is  $u = 5/13$  which gives  $L(u) = -46/13 = -3(7/13)$ .

## 12.3 EXTENSION TO INTEGER PROGRAMMING

Most of the results in this section are taken from Geoffrion [174]. See also Fisher [148]. Now assume that some or all of the variables in the linear program must take on integer values. The problem is

$$(MIP) \quad \begin{aligned} &\min c^\top x \\ &\text{s.t. } Ax \geq b \\ &\quad Bx \geq d \\ &\quad x \geq 0 \\ &\quad x_j \in \mathbb{Z}, j \in I \end{aligned}$$

where  $I$  is an index set contained in  $\{1, \dots, n\}$ . Throughout the rest of the chapter, define

$$\Gamma := \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathbb{Z}, j \in I\}. \quad (12.4)$$

In Chapter 4 we proved the following mixed integer finite basis theorem.

**Proposition 12.3 (Mixed Integer Finite Basis Theorem)** *If  $\Gamma$  is given by (12.4), then there exists  $x^1, \dots, x^r \in \Gamma$  such that*

$$\begin{aligned} \text{conv}(\Gamma) = \{x \in \Re^n \mid x = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i, \\ \sum_{i=1}^q z_i = 1, z_i \geq 0, i = 1, \dots, r\}. \end{aligned} \quad (12.5)$$

Now consider the *convex hull relaxation* of problem (MIP).

$$\begin{aligned} \text{conv}(MIP) \quad & \min c^\top x \\ & \text{s.t. } Ax \geq b \\ & \quad x \in \text{conv}(\Gamma) \end{aligned}$$

**Lemma 12.4** *Problem  $\text{conv}(MIP)$  is a relaxation of problem (MIP).*

Proof: Since  $\Gamma \subseteq \text{conv}(\Gamma)$  and both (MIP) and  $\text{conv}(MIP)$  have the same objective function  $c^\top x$  the definition of problem relaxation is satisfied.  $\square$

In Chapter 9 the linear programming relaxation of (MIP) was used to generate bounds for a branch-and-bound algorithm. The convex hull relaxation can also be used for bounding purposes in a branch and bound algorithm. The effectiveness of the convex hull relaxation depends on the quality of the bounds it provides and how difficult it is to optimize. The remainder of this section, and the following section, are devoted to demonstrating that the convex hull relaxation provides bounds equal to those of the Lagrangian dual and that these bounds are at least as tight, if not tighter, than the linear programming relaxation.

Substituting out  $x$  in  $\text{conv}(MIP)$  using (12.5) yields a formulation in the  $z$  equivalent to  $\text{conv}(MIP)$ .

$$\min \sum_{i=1}^r f_i z_i \quad (12.6)$$

$$\text{s.t. } \sum_{i=1}^r p^i z_i \geq b \quad (12.7)$$

$$\sum_{i=1}^q z_i = 1 \quad (12.8)$$

$$z_i \geq 0, \quad i = 1, \dots, r. \quad (12.9)$$

This is identical to problem (DW). Following the logic used in the previous section we show that the optimal value of the convex hull relaxation of (MIP) is equal to the optimal value of the Lagrangian dual of problem (MIP)

$$\max\{L(u) \mid u \geq 0\} \text{ where } L(u) = \min\{b^T u + (c - A^T u)^T x \mid x \in \Gamma\}.$$

**Proposition 12.5** *If  $\min\{c^T x \mid Ax \geq b, x \in \Gamma\}$  has an optimal solution, then*

$$\begin{aligned} & \min\{c^T x \mid Ax \geq b, x \in \text{conv}(\Gamma)\} \\ &= \min\left\{\sum_{i=1}^r f_i z_i \mid \sum_{i=1}^r p^i z_i \geq b, \sum_{i=1}^q z_i = 1, z_i \geq 0, i = 1, \dots, r\right\} \quad (12.10) \end{aligned}$$

$$\begin{aligned} &= \max\{b^T u + \min\{(f_i - u^T p^i) \mid i = 1, \dots, q\} \mid u \geq 0, \\ & \quad 0 \leq f_i - u^T p^i, i = q+1, \dots, r\} \quad (12.11) \end{aligned}$$

$$= \max\{\min\{b^T u + (c - A^T u)^T x \mid x \in \text{conv}(\Gamma)\} \mid u \geq 0\} \quad (12.12)$$

$$= \max\{\min\{b^T u + (c - A^T u)^T x \mid x \in \Gamma\} \mid u \geq 0\} \quad (12.13)$$

$$= \max\{L(u) \mid u \geq 0\} \quad (12.14)$$

**Proof:** Equation (12.10) follows from (12.5). Equation (12.11) follows from strong duality. Equation (12.12) follows from the fact that  $\text{conv}(\Gamma)$  is a polyhedron, the fundamental theorem of linear programming, and the hypothesis that (MIP) has an optimal solution. Equation (12.13) follows from the fact that the minimum of linear function over a set  $\Gamma$  is equal to the minimum of the linear function over  $\text{conv}(\Gamma)$ . Equation (12.14) is the definition of  $L(u)$ .

The following simple Corollary is left as an exercise. It is significant because it implies that the optimal Lagrangian dual variables can be found by solving a linear program. This is developed further in Section 12.5 on optimizing the dual function.

**Corollary 12.6** *If  $\min\{c^T x \mid Ax \geq b, x \in \Gamma\}$  where  $\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathbb{Z} \mid j \in I\}$ , has an optimal solution, then an optimal dual solution to the Lagrangian dual  $\max\{L(u) \mid u \geq 0\}$  is given by an optimal dual solution on the constraints  $\sum_{i=1}^r p^i z_i \geq b$  in the linear program (12.6)-(12.9).*

**Example 12.7 (Example 12.2 Continued)** *Assume that variables  $x_1$  and  $x_2$  in Example 12.2 must be integer. If  $\Gamma = \{(x_1, x_2) \mid 4x_1 + 9x_2 \geq 18, -2x_1 + 4x_2 \geq 4, x_1, x_2 \in \mathbb{Z}_+\}$ , then  $\text{conv}(\Gamma)$  is finitely generated by the four points  $(0,$*

$(0), (4,0), (2,1)$  and  $(0,1)$ . Then  $x \in \text{conv}(\Gamma)$  if and only if

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = z_1 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + z_2 \begin{bmatrix} 4 \\ 0 \end{bmatrix} + z_3 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + z_4 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where  $\sum_{i=1}^4 z_i = 1$  and  $z_1, z_2, z_3, z_4 \geq 0$ . The Dantzig-Wolfe master or the convex hull relaxation is

$$\begin{aligned} & \min 0z_1 - 4z_2 - 3z_3 - z_4 \\ \text{s.t. } & 0z_1 + 4z_2 + z_3 - z_4 \leq 2 \\ & z_1 + z_2 + z_3 + z_4 = 1 \\ & z_i \geq 0 \quad i = 1, \dots, 4 \end{aligned}$$

and the dual of this is

$$\begin{aligned} & \max -2u + u_0 \\ \text{s.t. } & u_0 \leq 0 \\ & u_0 \leq -4 + 4u \\ & u_0 \leq -3 + u \\ & u_0 \leq -1 - u. \end{aligned}$$

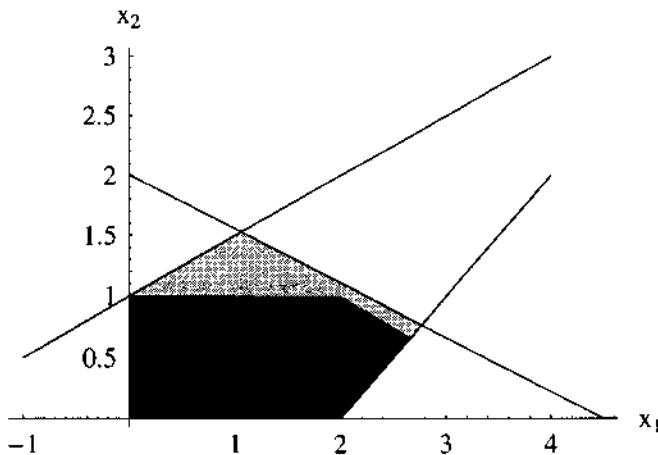
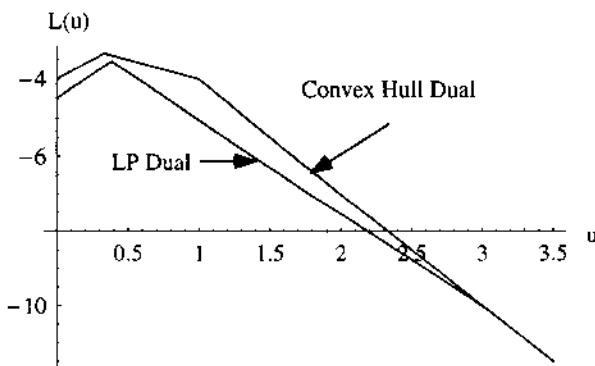
The Lagrangian function is

$$L(u) = -2u + \min\{0, -4 + 4u, -3 + u, -1 - u\}$$

and the optimal solution is  $u = 1/3$  and the optimal value of  $L(u) = -10/3 = -3(1/3)$ . This value is strictly greater than the value of  $-3(7/13)$  for the linear programming relaxation. In Figure 12.1 the darkly shaded region is  $\text{conv}(\Gamma)$ . The lightly shaded region is the additional area added by deleting the integer requirements on  $x_1$  and  $x_2$  in  $\Gamma$ . It is the feasible region of  $\bar{\Gamma}$  where

$$\bar{\Gamma} := \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0\}. \quad (12.15)$$

In Figure 12.2 two  $L(u)$  functions are plotted. The  $L(u)$  for Example 12.2 which is the linear programming version of the problem this problem and  $L(u)$  for the convex hull relaxation of the integer version of this problem. The convex hull relaxation is generating bounds as great, or greater, for all values of  $u$ . This interesting result is the topic of the next section. Of course the result is not surprising since adding the integer requirements on  $\Gamma$  reduces the set of feasible points in  $\text{conv}(\Gamma)$ .

**Figure 12.1**  $\text{conv}(\Gamma)$  for Example 12.7**Figure 12.2** The Lagrangian Dual Function for an Integer Program in Example 12.7 and its LP Relaxation

## 12.4 PROPERTIES OF THE LAGRANGIAN DUAL

The behavior of the dual function in Figure 12.2 is not a coincidence. It is a consequence of the following important Proposition.

**Proposition 12.8** If  $\min\{c^T x \mid Ax \geq b, x \in \Gamma\}$  has an optimal solution, then

$$\min\{c^T x \mid Ax \geq b, x \in \bar{\Gamma}\} \quad (12.16)$$

$$\leq \min\{c^T x \mid Ax \geq b, x \in \text{conv}(\Gamma)\} \quad (12.17)$$

$$= \max\{L(u) \mid u \geq 0\} \quad (12.18)$$

$$\leq \min\{c^T x \mid Ax \geq b, x \in \Gamma\} \quad (12.19)$$

**Proof:** Equation (12.17) is immediate from the fact that  $\text{conv}(\Gamma) \subseteq \bar{\Gamma}$ . Equation (12.18) is given in Proposition 12.5. Equation (12.19) follows from the fact that

$$(c - A^T u)^T \bar{x} + b^T u = u^T (b - A\bar{x}) + c^T \bar{x} \leq c^T \bar{x}$$

if  $A\bar{x} \geq b$  and  $u \geq 0$ . That is,

$$L(u) = \min\{(c - A^T u)^T x + b^T u \mid x \in \Gamma\}$$

is a relaxation of  $\min\{c^T x \mid Ax \geq b, x \in \Gamma\}$  for any nonnegative  $u$ .  $\square$

This technique of forming the Lagrangian dual is often called *Lagrangian relaxation*. Proposition 12.8 is perhaps the most significant proposition in this chapter. By (12.18)-(12.19),

$$L(u) \leq \min\{c^T x \mid Ax \geq b, x \in \Gamma\}$$

so for any nonnegative  $u$ ,  $L(u)$  provides a lower bound on  $\min\{c^T x \mid Ax \geq b, x \in \Gamma\}$ . Indeed the optimization problem

$$L(u) = \min\{(c - A^T u)^T x + b^T u \mid x \in \Gamma\}$$

is a relaxation of the integer program

$$\min\{c^T x \mid Ax \geq b, x \in \Gamma\}$$

and this technique is often called *Lagrangian relaxation*. This is also a *weak duality* result for integer linear programming. See Lemma 2.28 in Chapter 2 for the linear programming weak duality result. From equation (12.17) the Lagrangian dual bounds are potentially stronger than the linear programming relaxation studied earlier. This has significance if the Lagrangian dual is incorporated within an enumeration scheme.

If

$$\min\{b^T u + (c - A^T u)^T x \mid x \in \Gamma\} = \min\{b^T u + (c - A^T u)^T x \mid x \in \bar{\Gamma}\}$$

for all  $u \geq 0$ , then the Lagrangian dual for (MIP) is said to have the *integrality property*. See Geoffrion [174]. As the following Corollary shows, the integrality property is, in a sense, “bad” since in this case the Lagrangian dual does not provide bounds that are stronger than the linear programming relaxation.

**Corollary 12.9 (Integrality Property)** *If the integrality property holds for the mixed integer linear programming problem,  $\min\{c^\top x \mid Ax \geq b, x \in \Gamma\}$  when the constraints  $Ax \geq b$  are dualized, and the linear program  $\min\{c^\top x \mid Ax \geq b, x \in \bar{\Gamma}\}$  has an optimal solution, then*

$$\min\{c^\top x \mid Ax \geq b, x \in \bar{\Gamma}\} = \max\{L(u) \mid u \geq 0\}$$

where  $L(u) = \min\{u^\top b + (c - A^\top u)^\top x \mid x \in \Gamma\}$ .

Not only does  $L(u)$  provide a new lower bound on the optimal value of problem (MIP), but this function has special structure. In the next section we show how to exploit this special structure in order to generate bounds quickly.

**Proposition 12.10** *For problem (MIP)  $\min\{c^\top x \mid Ax \geq b, x \in \Gamma\}$  where  $\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \text{ integer}, j \in I\}$ , the Lagrangian dual function  $L(u)$  is piecewise linear and concave.*

**Proof:** The result is immediate from Equations (12.11)-(12.14) which imply that  $L(u)$  is the piecewise minimum of linear functions defined on a convex set.  $\square$

From Proposition 12.8,  $\max\{L(u) \mid u \geq 0\} \leq \min\{c^\top x \mid Ax \geq b, x \in \Gamma\}$ , which is a weak duality result for mixed integer linear programming. In Proposition 12.1 we gave a strong duality result for the Lagrangian dual when  $\Gamma$  is a polyhedron. It is natural to attempt to extend this result to a strong duality result, or at least find sufficient conditions for the optimal value of the Lagrangian dual to equal the optimal value of the primal when  $\Gamma$  is not a polyhedron.

A point  $(\bar{x}, \bar{u})$  with  $\bar{u} \geq 0$  and  $\bar{x} \in \Gamma$  is a *saddlepoint* for  $L(x, u) := b^\top u + (c - Au^\top)^\top x$  if and only if

$$\max_{u \geq 0} L(\bar{x}, u) \leq L(\bar{x}, \bar{u}) \leq \min_{x \in \Gamma} L(x, \bar{u}). \quad (12.20)$$

A more useful characterization of a saddlepoint from a computational standpoint is given in Proposition 12.11 below.

**Proposition 12.11** If  $\bar{u} \geq 0$  and  $\bar{x} \in \Gamma$ , then  $(\bar{x}, \bar{u})$  is a saddlepoint for  $L(x, u) = b^T u + (c - A^T u)^T x$  if and only if:

$$\bar{x} \text{ minimizes } L(x, \bar{u}) \text{ over } \Gamma, \quad (12.21)$$

$$A\bar{x} \geq b, \quad (12.22)$$

$$\bar{u}^T (b - A\bar{x}) = 0. \quad (12.23)$$

**Proof:** Assume conditions (12.21)-(12.23) hold for  $\bar{u} \geq 0$  and  $\bar{x} \in \Gamma$ . By condition (12.21),  $L(\bar{x}, \bar{u}) \leq \min_{x \in \Gamma} L(x, \bar{u})$ . By condition (12.23),  $L(\bar{x}, \bar{u}) = c^T \bar{x}$ . By (12.22),  $A\bar{x} \geq b$ , so for all  $u \geq 0$ ,

$$L(\bar{x}, u) = c^T \bar{x} + u^T (b - A\bar{x}) \leq c^T \bar{x} = L(\bar{x}, \bar{u}).$$

Therefore,  $\max_{u \geq 0} L(\bar{x}, u) \leq L(\bar{x}, \bar{u})$ .

Now assume  $\bar{x} \in \Gamma$ ,  $\bar{u} \geq 0$  and that  $(\bar{x}, \bar{u})$  is a saddlepoint of  $L(x, u)$ . By definition of saddlepoint,  $L(\bar{x}, \bar{u}) \leq \min_{x \in \Gamma} L(x, \bar{u})$ . Thus  $\bar{x}$  is an optimal solution to the problem  $\min_{x \in \Gamma} L(x, \bar{u})$  and condition (12.21) is satisfied. Since  $\max_{u \geq 0} L(\bar{x}, u) \leq L(\bar{x}, \bar{u})$  every component of  $A\bar{x} - b$  must be nonnegative, otherwise  $\max_{u \geq 0} L(\bar{x}, u) \leq$  is unbounded. Thus, condition (12.22) is satisfied. Finally,  $(a_i)^T \bar{x} > b_i$  implies  $\bar{u}_i = 0$  since  $\bar{u}$  is an optimal solution to  $\max_{u \geq 0} L(\bar{x}, u)$  and condition (12.23) is satisfied.  $\square$

The following Proposition, which is left as an exercise, shows that if we can find a saddlepoint then we have an optimal primal-dual pair.

**Proposition 12.12** The pair  $(\bar{x}, \bar{u})$  is a saddlepoint for  $L(x, u)$  if and only if  $\bar{x}$  is primal feasible,  $\bar{u}$  is dual feasible, and  $c^T \bar{x} = L(\bar{u})$ .

Conditions (12.21)-(12.23) are the *optimality conditions* for problem (MIP). This is because any primal-dual pair  $(\bar{x}, \bar{u})$  which satisfy the optimality conditions are an optimal primal-dual pair. These optimality conditions generalize to integer linear programming the optimality conditions given in Corollary 2.34 in Chapter 2. Unfortunately, in the case of integer linear programming the optimality conditions are only sufficient but not necessary. The difference between the optimal value of MIP and the optimal value of the Lagrangian dual  $L(u)$  is referred to as the *duality gap*. Whenever there is a duality gap it is not possible to satisfy the optimality conditions (i.e. find a saddle point). When using the term "duality gap" it is important to know the context, i.e. which dual problem is used. When the dual of the integer program is the dual of the linear programming relaxation then the duality gap is the same as the integrality gap.

Many algorithms work by optimizing the Lagrangian dual and then finding a complementary feasible primal solution. However, for some problems (e.g., integer programs) we may optimize the Lagrangian dual but not obtain a complementary primal feasible solution as a result. However, if a problem, such as an integer program, has a duality gap, solving the Lagrangian problem  $\min_{x \in \Gamma} L(x, u)$  still provides interesting information. We motivate the concept first by an example.

### Example 12.13

$$\begin{aligned} & \min 40x_1 + 60x_2 + 70x_3 + 160x_4 \\ & 16x_1 + 35x_2 + 45x_3 + 85x_4 \geq 81 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\} \end{aligned}$$

This is a binary knapsack problem. This problem structure arises from capital budgeting problems. See Subsection 1.3.2 in Chapter 1 and the work of Lorie and Savage [293]. In this application setting variable  $x_i = 1$  if project  $i$  is undertaken and  $x_i = 0$  if project  $i$  is not undertaken. The coefficients of  $x_i$  in the objective function represent the cost of each project. The coefficients in the knapsack constraint represent the capital return of each of the projects. The right hand side of 81 on the knapsack constraint represents the capital return that must be achieved. Dualizing the budget constraint gives

$$\begin{aligned} L(u) &= \min \{81u + (40x_1 + 60x_2 + 70x_3 + 160x_4 - \\ & u(16x_1 + 35x_2 + 45x_3 + 85x_4)) \mid x_i \in \{0, 1\}, i = 1, \dots, 4\} \\ &= \min \{81u + (40 - 16u)x_1 + (60 - 35u)x_2 + (70 - 45u)x_3 \\ & \quad + (160 - 85u)x_4 \mid x_i \in \{0, 1\}, i = 1, \dots, 4\}. \end{aligned}$$

For any  $u \geq 0$ ,  $L(u)$  is very easy to optimize. Since  $\Gamma = \{x \mid x_i \in \{0, 1\}\}$ , for any  $u = \bar{u}$ , we just “peg”  $x_i = 1$  if  $(c_i - \bar{u}a_i) < 0$  and  $x_i = 0$  if  $(c_i - \bar{u}a_i) \geq 0$ . Then the optimal value of  $L(u)$  for any  $u = \bar{u}$  is

$$L(\bar{u}) = \bar{u}b + \sum_{i=1}^n \min\{0, (c_i - \bar{u}a_i)\}.$$

There are alternative optima when  $(c_i - \bar{u}a_i) = 0$  since we can peg  $x_i$  to either 0 or 1. For example, if  $\bar{u} = 2$  then

$$L(2) = \min\{162 + 8x_1 - 10x_2 - 20x_3 - 10x_4 \mid x_i \in \{0, 1\}, i = 1, \dots, 4\} = 162 - 40 = 122.$$

Letting  $u$  vary over the nonnegative orthant gives the following family of solutions (the alternative optimal solution for the right endpoint of the interval is

also given).

	$x_1$	$x_2$	$x_3$	$x_4$	$x_1$	$x_2$	$x_3$	$x_4$
$0 \leq u \leq 14/9$	0	0	0	0	0	0	1	0
$14/9 < u \leq 12/7$	0	0	1	0	0	1	1	0
$12/7 < u \leq 32/17$	0	1	1	0	0	1	1	1
$32/17 < u \leq 2.5$	0	1	1	1	1	1	1	1
$2.5 < u$	1	1	1	1	1	1	1	1

This gives the following  $L(u)$ ,

$$L(u) = \begin{cases} 81u, & 0 \leq u \leq 14/9 \\ 70 + 36u, & 14/9 < u \leq 12/7 \\ 130 + u, & 12/7 < u \leq 32/17 \\ 290 - 84u, & 32/17 < u \leq 2.5 \\ 330 - 100u, & 2.5 < u \end{cases}$$

The optimal value of  $u$  is  $\bar{u} = 32/17$  and  $L(32/17) = 131 + (15/17)$ . The optimal primal solution is  $\bar{x}_1 = 0$ ,  $\bar{x}_2 = 0$ ,  $\bar{x}_3 = 0$ , and  $\bar{x}_4 = 1$ . The optimal primal solution value is 160. Thus, there is a substantial duality gap. The optimal primal dual solution fails to satisfy the complementary slackness condition (12.23) of the optimality conditions because there is positive slack of 1 unit on the knapsack constraint and it has a positive optimal dual variable. However, if we required a capital return of only 80 units, instead of 81 units, then  $\bar{x}_1 = 0$ ,  $\bar{x}_2 = 1$ ,  $\bar{x}_3 = 1$ , and  $\bar{x}_4 = 0$  and  $\bar{u} = 32/17$ , constitute an optimal primal-dual solution since the optimality conditions (12.21)-(12.23) are satisfied. By requiring one less unit of return we can save  $(160 - 130) = 30$  cost units!

The following important Propositions are due to Everett [145].

**Proposition 12.14 (Everett)** If, for  $\bar{u} \geq 0$ ,  $\bar{x}$  is an optimal solution to  $L(\bar{u}) = \min\{b^\top \bar{u} + (c - A^\top \bar{u})^\top x \mid x \in \Gamma\}$ , then  $\bar{x}$  is an optimal solution of the modified primal problem

$$\min\{c^\top x \mid Ax \geq \hat{b}, x \in \Gamma\}$$

where  $\hat{b}_i = (a^i)^\top \bar{x}$  for all  $\bar{u}_i > 0$  and  $\hat{b}_i \leq (a^i)^\top \bar{x}$  for all  $\bar{u}_i = 0$ .

**Proof:** Simply observe that  $(\bar{x}, \bar{u})$  is a saddle point for

$$L(x, u) = c^\top x + u^\top (\hat{b} - Ax). \square$$

This is a very nice result because  $\hat{b}$  may be close enough to  $b$  so that, considering the accuracy of the data or the real problem situation, we have really optimized the primal problem for all intents and purposes. This was illustrated in Example 12.13 where allowing a perturbation of the right hand side by 1 unit saved 30 units in the primal objective function value.

**Proposition 12.15** *If  $\bar{u} \geq 0$ ,  $\bar{x}$  is an optimal solution to  $L(\bar{u}) = \min\{b^T \bar{u} + (c - A^T \bar{u})^T x \mid x \in \Gamma\}$  and  $A\bar{x} \geq b$ , then*

$$c^T \bar{x} - \min\{c^T x \mid Ax \geq b, x \in \Gamma\} \leq \bar{u}^T (b - A\bar{x}).$$

**Proof:** By weak duality,  $\min\{c^T x \mid Ax \geq b, x \in \Gamma\} \geq L(\bar{u})$  so,

$$\begin{aligned} c^T \bar{x} - \min\{c^T x \mid Ax \geq b, x \in \Gamma\} \\ \leq c^T \bar{x} - L(\bar{u}) \\ = \bar{u}^T (b - A\bar{x}) \quad \square \end{aligned}$$

Proposition 12.15 says that if we meet optimality conditions (12.21) and (12.22) then we have a primal feasible solution and the amount by which we miss complementary slackness provides a bound on how far we are away from the optimal primal solution value. If  $\bar{u}^T (b - A\bar{x})$  were sufficiently small one might be satisfied to stop with  $\bar{x}$  on the grounds that it is “close enough” to the optimal solution!

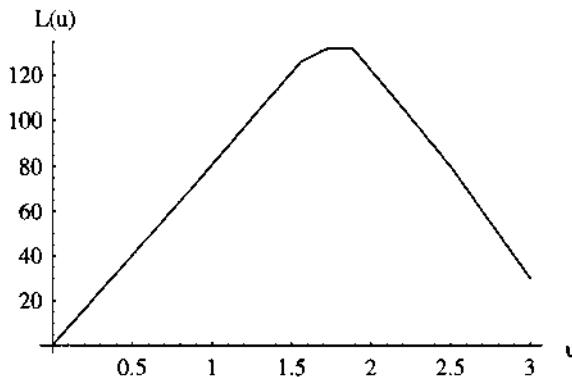
What is the upshot of all this? Well, we may be given a very difficult primal problem to solve, maybe an integer linear program. By judiciously determining what the  $Ax \geq b$  and  $\Gamma$  constraints are, we may get a Lagrangian problem which is relatively easy to solve (at least compared with the original primal). By applying Proposition 12.14 and Proposition 12.15 we can find optimal solutions to closely “related problems” and possibly determine tight bounds on the original primal solution value. Of course this procedure requires optimizing  $L(u)$  which is the topic of the next section.

## 12.5 OPTIMIZING THE LAGRANGIAN DUAL

Refer to the capital budgeting problem in Example 12.13. The Lagrangian function for this problem is plotted in Figure 12.3. This Figure illustrates three interesting features of  $L(u)$ .

1. The Lagrangian function  $L(u)$  is piecewise linear and therefore differentiable except for a finite set of  $u$  values.
2. The function is not differentiable at the  $u$  values where  $L(u) = \min_{x \in \Gamma} L(x, u)$  has alternative optima.
3. If  $L(u)$  is differentiable for  $u = \bar{u}$ , and  $\bar{x}$  is the vector which minimizes  $L(x, \bar{u})$ , then the derivative of  $L(u)$  is  $(81 - 16\bar{x}_1 - 35\bar{x}_2 - 45\bar{x}_3 - 85\bar{x}_4)$ . When  $L(u)$  is not differentiable for  $u = \bar{u}$ , the alternative optima which minimize  $L(x, \bar{u})$ , when evaluated in the knapsack constraint  $b - a^T x$  give the left and right derivatives.

**Figure 12.3** The Lagrangian Dual Function for Capital Budgeting Example



How do these observations generalize? A vector,  $\gamma \in \mathbb{R}^m$ , is a *subgradient* of  $L(u)$  at  $\bar{u}$  if and only if

$$L(u) \leq L(\bar{u}) + (u - \bar{u})^T \gamma, \quad u \in R^m. \quad (12.24)$$

Geometrically, what is  $L(\bar{u}) + (u - \bar{u})^T \gamma$ ? Let  $\Gamma(\bar{u})$  denote the set of optimal solutions for the Lagrangian function at  $\bar{u}$ , that is,  $\bar{x} \in \Gamma(\bar{u})$  if and only if

$$L(\bar{u}) = c^T \bar{x} + \bar{u}^T (b - A\bar{x}) = \min\{b^T \bar{u} + (c - A^T \bar{u})^T x \mid x \in \Gamma\}.$$

**Proposition 12.16** *If  $\bar{x} \in \Gamma(\bar{u})$ , then the vector  $\bar{\gamma} = (b - A\bar{x})$  is a subgradient of  $L(u)$  at  $\bar{u}$ .*

**Proof:** For any  $u$ , by definition of  $L$ ,

$$\begin{aligned} L(u) &\leq c^\top \bar{x} + u^\top (b - A\bar{x}) \\ &= L(\bar{u}) - \bar{u}^\top (b - A\bar{x}) + u^\top (b - A\bar{x}) \\ &= L(\bar{u}) + (u - \bar{u})^\top (b - A\bar{x}) \end{aligned}$$

Let  $\gamma = (b - A\bar{x})$  and this completes the proof.  $\square$

**Corollary 12.17** *The dual objective  $L(u)$  is differentiable at a point  $\bar{u}$  if and only if  $b_i - (a^i)^\top x$  is constant over  $\Gamma(\bar{u})$  for all  $i$ . In this case, the gradient of  $L(\bar{u})$  is given by  $(b - A\bar{x})$  for all  $\bar{x} \in \Gamma(\bar{u})$ .*

The subgradient extends the concept of a gradient to nondifferentiable concave/convex functions. We have seen the importance of the dual problem and some of its properties. Now let's see how we can optimize the dual function.

### 12.5.1 Method 1: Subgradient Optimization

Subgradient optimization extends the method of steepest ascent to the dual function which is concave and piecewise linear (see Proposition 12.10). See a nonlinear programming text such as Bazaraa, Sherali, and [44] for a treatment of gradients and steepest ascent algorithms. Unfortunately, moving in the direction of a subgradient does not necessarily lead to an improvement in the dual function. It is left as an exercise to construct an example problem where moving in the direction of a subgradient does not necessarily lead to an improvement in the dual objective function value. However, there is a rational for moving in the direction of a subgradient.

Assume  $\bar{u}$  is an optimal solution to the dual problem  $\max_{u \geq 0} L(u)$ . If  $u^j \geq 0$  and  $\gamma^j = g(x^j)$  where  $x^j \in \Gamma(u^j)$  it follows from the definition of subgradient that

$$L(\bar{u}) \leq L(u^j) + (\bar{u} - u^j)^\top \gamma^j$$

which implies

$$L(\bar{u}) - L(u^j) \leq (\bar{u} - u^j)^\top \gamma^j$$

But  $\bar{u}$  is an optimal dual solution so  $L(\bar{u}) - L(u^j) \geq 0$  which implies  $(\bar{u} - u^j)^\top \gamma^j \geq 0$ . Then  $\gamma^j$  makes an acute angle with the ray from  $u^j$  through

$\bar{u}$ . Let  $u^{j+1} = \max\{0, u^j + t_j \gamma^j\}$  where  $t_j$  is a positive scalar called the step size. Consequently, if  $t_j$  is sufficiently small, the point  $u^{j+1}$  will be closer to  $\bar{u}$  than  $u^j$ . Thus, the sequence of dual solutions generated by the subgradient algorithm approaches an optimal solution, although the value of  $L(u^{j+1})$  is not necessarily greater than  $L(u^j)$ .

**Algorithm 12.18 (Subgradient Optimization Algorithm)**

**Step 1: (Initialization)** Let  $j \leftarrow 0$ ,  $u^j \in (R^m)^+$  and  $\epsilon > 0$ .

**Step 2:**  $\gamma^j \leftarrow g(x^j)$  where  $x^j \in \Gamma(u^j)$ .

**Step 3:** Let  $u^{j+1} \leftarrow \max\{0, u^j + t_j \gamma^j\}$  where  $t_j$  is a positive scalar called the step size.

**Step 4:** If  $\|u^{j+1} - u^j\| < \epsilon$  stop else let  $j \leftarrow j + 1$  and go to **Step 2**.

The question that remains is how to select the  $t_j$  in order to guarantee convergence. The fundamental theoretical result (see Held, Wolfe, and Crowder [226] and references therein) is that the sequence  $\{L(u^j)\}$  converges to  $L(\bar{u})$  if the sequence  $\{t_j\}$  converges to zero and  $\sum_{j=0}^{\infty} t_j = \infty$ . The reason for the condition  $\sum_{j=0}^{\infty} t_j = \infty$  is to make the step sizes “large enough” to get from an initial  $u^0$  to an optimal  $\bar{u}$ . However, we want the  $t_j \rightarrow 0$  so we can’t continually “overshoot”  $\bar{u}$ . It is common to determine the  $t_j$  by a formula such as

$$t_j = \frac{\theta_j(L^* - L(u^j))}{\|\gamma^j\|^2} \quad (12.25)$$

where  $\theta_j$  is a positive scalar between 0 and 2. Often the  $\theta_j$  are determined by setting  $\theta_0 = 2$  and halving  $\theta_j$  whenever  $L(u)$  has failed to increase for some fixed number of iterations. The term  $L^*$  is an upper bound on the optimal value  $L(u)$ . By weak duality, one such valid bound is the value of any primal feasible solution. Getting a subgradient algorithm to work in practice requires a considerable amount of “tweaking” the parameters. The best parameter values are very problem specific.

**Example 12.19 (The Generalized Assignment Problem)** The generalized assignment problem has the following generic form.

$$\min \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} \quad (12.26)$$

$$(GAP) \quad \text{s.t.} \quad \sum_{j=1}^m x_{ij} = 1, \quad i = 1, \dots, n \quad (12.27)$$

$$\sum_{i=1}^n a_{ij} x_{ij} \leq b_j, \quad j = 1, \dots, m \quad (12.28)$$

$$x_{ij} \in \{0, 1\} \quad (12.29)$$

The problem was first introduced in Subsection 1.3.3 in Chapter 1. In this formulation  $x_{ij} = 1$  if “task”  $i$  is assigned to “server”  $j$ , 0, otherwise. In the objective function (12.26)  $c_{ij}$  is the cost of assigning task  $i$  to server  $j$ . Constraint set (12.27) requires that every task is assigned exactly one server. If server  $j$  performs task  $i$ ,  $a_{ij}$  units of resource are consumed and server  $j$  has  $b_j$  units of this resource available. Constraint set (12.28) does not allow server  $j$  to exceed the resource available. The binary requirements (12.29) do not allow any fractional assignments. See Fisher and Jaikumar [149] for an excellent description of how the generalized assignment problem arises as a subproblem in capacitated vehicle routing problems.

In applying Lagrangian relaxation to a problem it is necessary to figure out which constraints to relax (i.e. dualize). In the generalized assignment problem this requires choosing between the assignment constraints (12.27) or the resource limitation constraints (12.28). If the resource limitation constraints are relaxed, the resulting optimization problem is

$$\begin{aligned} L(u) &= \min \quad \sum_{i=1}^n \sum_{j=1}^m (c_{ij} + u_j a_{ij}) x_{ij} - \sum_{j=1}^m u_j b_j \\ \text{s.t.} \quad &\sum_{j=1}^m x_{ij} = 1, \quad i = 1, \dots, n \\ &x_{ij} \in \{0, 1\} \end{aligned}$$

It is easy to show that the Lagrangian function  $L(u)$  has the integrality property and for the optimal  $u$ ,  $L(u)$  will equal the linear programming relaxation value of (GAP). The linear programming relaxation for generalized assignment problems is usually a very poor approximation for the binary solution. It is usually much better to relax the assignment constraints so that the resulting dual function is

$$\begin{aligned} L(u) &= \min \quad \sum_{i=1}^n \sum_{j=1}^m (c_{ij} - u_i) x_{ij} + \sum_{i=1}^n u_i \\ \text{s.t.} \quad &\sum_{i=1}^n a_{ij} x_{ij} \leq b_j, \quad j = 1, \dots, m \\ &x_{ij} \in \{0, 1\} \end{aligned}$$

This Lagrangian function has considerable special structure. Even though it does not have the integrality property and is an integer program, it separates into

*m* distinct knapsack problems, one for each server. There are numerous special purpose algorithms for solving knapsack problems. See the book by Martello and Toth [313]. Since  $L(u)$  is optimized relatively easily given a  $u$  vector, this Lagrangian function is a good candidate for subgradient optimization. We illustrate the process on the specific example below.

$$\begin{aligned} \min \quad & 9x_{11} + 2x_{12} + x_{21} + 2x_{22} + 3x_{31} + 8x_{32} \\ \text{s.t.} \quad & x_{11} + x_{12} = 1 \\ & x_{21} + x_{22} = 1 \\ & x_{31} + x_{32} = 1 \\ & 6x_{11} + 7x_{21} + 9x_{31} \leq 13 \\ & 8x_{12} + 5x_{22} + 6x_{32} \leq 11 \\ & x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in \{0, 1\} \end{aligned}$$

The linear programming relaxation solution for this example is given below.

OBJECTIVE VALUE = 6.42857120

#### OBJECTIVE FUNCTION VALUE

1) 6.428571

VARIABLE	VALUE	REDUCED COST
X12	1.000000	.000000
X21	.571429	.000000
X22	.428571	.000000
X31	1.000000	-3.714286

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-2.000000
3)	.000000	-2.000000
4)	.000000	-8.000000
5)	.000000	.142857

Let  $u^0$  be the optimal linear programming relaxation dual variable values on the assignment constraints. Then

$$\begin{aligned} L(u^0) = \min \quad & 7x_{11} + 0x_{12} - x_{21} + 0x_{22} - 5x_{31} + 0x_{32} + 12 \\ \text{s.t.} \quad & 6x_{11} + 7x_{21} + 9x_{31} \leq 13 \\ & 8x_{12} + 5x_{22} + 6x_{32} \leq 11 \\ & x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in \{0, 1\} \end{aligned}$$

The optimal solution to this subproblem is  $x_{31}^0 = 1$  and all other variables equal to zero. This gives  $L(u^0) = 7$ . This is strictly greater than the linear programming relaxation value of 6.4285712 and illustrates Proposition 12.8. By Proposition 12.16 the subgradient for  $u^0$  is

$$\begin{aligned}\gamma_1^0 &= 1 - x_{11}^0 - x_{12}^0 = 1 \\ \gamma_2^0 &= 1 - x_{21}^0 - x_{22}^0 = 1 \\ \gamma_3^0 &= 1 - x_{31}^0 - x_{32}^0 = 0\end{aligned}$$

In order to proceed with the subgradient algorithm a stepsize is required. Use the value given in (12.25). An upper bound  $L^*$ , on the optimal dual value is also needed. By weak duality any feasible primal solution value is valid. An obvious feasible solution is  $x_{11} = 1$ ,  $x_{21} = 1$  and  $x_{32} = 1$  for  $L^* = 19$ . Start with  $\theta = 1$  in equation (12.25) and get  $t_0 = 1(19 - 7)/2 = 6$ . Then from Step 3 of the subgradient algorithm,  $u_1^1 \leftarrow u_1^0 + 6(1)$ ,  $u_2^1 \leftarrow u_2^0 + 6(1)$ ,  $u_3^1 \leftarrow u_3^0 + 6(0)$ . The new dual vector is  $u^1 = (8, 8, 8)$ . Then,

$$\begin{aligned}L(u^1) = \min \quad & x_{11} - 6x_{12} - 7x_{21} - 6x_{22} - 5x_{31} + 0x_{32} + 24 \\ \text{s.t.} \quad & 6x_{11} + 7x_{21} + 9x_{31} \leq 13 \\ & 8x_{12} + 5x_{22} + 6x_{32} \leq 11 \\ & x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in \{0, 1\}\end{aligned}$$

The optimal solution to this subproblem is  $x_{21} = 1$ ,  $x_{22} = 1$ , and all others equal zero. This gives  $L(u^1) = 11$ . The new subgradient is

$$\begin{aligned}\gamma_1^0 &= 1 - x_{11}^0 - x_{12}^0 = 1 \\ \gamma_2^0 &= 1 - x_{21}^0 - x_{22}^0 = -1 \\ \gamma_3^0 &= 1 - x_{31}^0 - x_{32}^0 = 1\end{aligned}$$

The new step size is  $t_1 = 1(19 - 11)/3 = 8/3$ . Then from Step 3 of the subgradient algorithm,  $u_1^2 \leftarrow u_1^1 + \frac{8}{3}(1)$ ,  $u_2^2 \leftarrow u_2^1 + \frac{8}{3}(-1)$ ,  $u_3^2 \leftarrow u_3^1 + \frac{8}{3}(1)$ . The new dual vector is  $u^2 = (10\frac{2}{3}, 5\frac{1}{3}, 10\frac{2}{3})$ .

$$\begin{aligned}L(u^2) = \min \quad & -1\frac{2}{3}x_{11} - 8\frac{2}{3}x_{12} - 4\frac{1}{3}x_{21} - 3\frac{1}{3}x_{22} - 7\frac{2}{3}x_{31} - 2\frac{2}{3}x_{32} + 26\frac{2}{3} \\ \text{s.t.} \quad & 6x_{11} + 7x_{21} + 9x_{31} \leq 13 \\ & 8x_{12} + 5x_{22} + 6x_{32} \leq 11 \\ & x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in \{0, 1\}\end{aligned}$$

The optimal solution is  $x_{31} = 1$ ,  $x_{12} = 1$ , and all others equal zero. This gives a dual value of  $10\frac{1}{3}$  which is worse than the previous value of 11. One could either calculate a new subgradient and move in that direction, or halve the value of  $\theta$  used to calculate  $t_1$  and try  $t_1 = 4/3$ . Try halving the value of  $\theta$ . This gives

$$u^2 = (9\frac{1}{3}, 6\frac{2}{3}, 9\frac{1}{3}).$$

$$\begin{aligned} L(u^2) = \min \quad & -\frac{1}{3}x_{11} - 7\frac{1}{3}x_{12} - 5\frac{2}{3}x_{21} - 4\frac{2}{3}x_{22} - 6\frac{1}{3}x_{31} - 1\frac{1}{3}x_{32} + 25\frac{1}{3} \\ \text{s.t.} \quad & 6x_{11} + 7x_{21} + 9x_{31} \leq 13 \\ & 8x_{12} + 5x_{22} + 6x_{32} \leq 11 \\ & x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in \{0, 1\} \end{aligned}$$

The optimal solution is  $x_{31} = 1$ ,  $x_{12} = 1$ , and all others equal zero. This gives a dual value of  $11\frac{2}{3}$ . The solution is converging to the optimal dual solution of 12.5 and  $\bar{u} = (9, 7.5, 9.5)$ .

The technique of subgradient optimization has worked well in practice. One of the first applications was by Held and Karp [225] who used subgradient optimization to generate bounds on a relaxation of the traveling salesman problem. Subgradient optimization has also been applied successfully to the “lock box” or simple plant location problem. See Cornuéjols, Fisher, and Nemhauser [101]. Although the integrality property holds for simple plant location problems when the demand constraints are dualized, the Lagrangian subproblem has so much special structure that subgradient optimization proved much faster than the simplex algorithm. Afentakis, Gavish, and Karmarkar [5] used subgradient optimization to optimize the Lagrangian dual of multiproduct, multiperiod lot sizing problems. This application is developed in detail in the next subsection.

### 12.5.2 Method 2: Column Generation

By Corollary 12.6 an optimal dual solution  $\bar{u}$  for the Lagrangian dual problem  $\max\{L(u) | u \geq 0\}$  corresponds to an optimal dual solution for the constraint set  $\sum_{i=1}^T p^i z_i \geq b$  in the linear programming formulation of the convex hull relaxation of problem (MIP) derived using the mixed integer finite basis theorem. See formulation (12.6)-(12.9). The method for solving this linear program is identical to the method outlined in the Chapter 11 on Dantzig-Wolfe decomposition. A small subset of the columns in (12.6)-(12.9) are kept and columns are added on the fly. The subproblem used to add the columns is the Lagrangian subproblem  $\min\{L(x, u) | x \in \Gamma\}$ . This procedure is illustrated in the following example.

**Example 12.20 (Multiproduct Dynamic Lot Size)** In this example we consider the classic multiproduct, multiperiod dynamic lot sizing problem. This problem was introduced in Subsection 1.3.4 in Chapter 1. This problem, and

variations of it, have been studied by Afentakis, Gavish, and Karmarkar [5], Barany, Van Roy, and Wolsey [34], Graves [206], Dzielinski and Gomory [133], Lasdon and Terjung [278], Manne [308], and many others. Recall that the parameters are  $T$ , the number of time periods;  $N$ , the number of products;  $d_{it}$ , the demand for product  $i$  in period  $t$ ;  $f_{it}$ , the fixed cost associated with production of product  $i$  in period  $t$ ;  $h_{it}$ , the marginal cost of holding one unit of product  $i$  in inventory at the end of period  $t$ ;  $c_{it}$ , the marginal production cost of one unit of product  $i$  in period  $t$ ; and  $g_t$ , the production capacity available in period  $t$ . The variables are  $x_{it}$ , the units of product  $i$  produced in period  $t$ ;  $I_{it}$ , the units of product  $i$  held in inventory at the end of period  $t$ ; and  $y_{it}$ , a binary variable which is fixed to 1 if there is nonzero production of product  $i$  in period  $t$ , otherwise it is fixed to 0.

The multiproduct dynamic lot size mixed integer program is

$$\min \quad \sum_{i=1}^N \sum_{t=1}^T (c_{it}x_{it} + h_{it}I_{it} + f_{it}y_{it}) \quad (12.30)$$

$$(MPDLS) \quad \text{s.t.} \quad \sum_{i=1}^N x_{it} \leq g_t, \quad t = 1, \dots, T \quad (12.31)$$

$$I_{i,t-1} + x_{it} - I_{it} = d_{it}, \quad i = 1, \dots, N, \quad t = 1, \dots, T \quad (12.32)$$

$$x_{it} - M_{it}y_{it} \leq 0, \quad i = 1, \dots, N, \quad t = 1, \dots, T \quad (12.33)$$

$$x_{it}, I_{it} \geq 0, \quad i = 1, \dots, N, \quad t = 1, \dots, T \quad (12.34)$$

$$y_{it} \in \{0, 1\}, \quad i = 1, \dots, N, \quad t = 1, \dots, T. \quad (12.35)$$

Constraints (12.31) are the production capacity constraints (and will be treated as the coupling constraints), constraints (12.32) are the demand balance constraints and constraints (12.33) are the setup forcing constraints. The basic idea here is to force variable  $y_{it}$  to be 1 if  $x_{it} > 0$ . To this end, the "big  $M_{it}$ " must be selected large enough so as not to remove any feasible production solutions. A valid choice for  $M_{it}$  is  $M_{it} = \sum_{t=1}^T d_{it}$ . The importance of big  $M_{it}$  is discussed later in Section 15.1 in Chapter 15.

A realistic problem might involve several hundred products and 10 to 20 time periods. This does not result in an unreasonably large linear program (several thousand rows and variables are not a problem), however the  $y_{it}$  must be 0/1 and even selecting the smallest possible  $M_{it}$  results in a mixed 0/1 linear program that is impossible to solve with any commercially available code. The linear programming relaxation bounds are simply too weak and too much enumeration is required. Therefore, let's attack this with a decomposition-column generation approach and generate tight bounds using the Lagrangian dual. Constraints (12.31)-(12.35) are block angular where there is a block for each product, and

Table 12.1 Two Product, Single Machine, Dynamic Lotsize Example

<u>Demand</u>	<u>May</u>	<u>June</u>	<u>July</u>	<u>August</u>	<u>September</u>
Product 1	60	100	140	200	120
Product 2	40	60	100	40	80
<u>Setup Cost</u>					
Product 1	150	140	160	160	160
Product 2	30	30	40	50	35
<u>Variable Cost</u>					
Product 1	7	7	8	7	7
Product 2	2	4	4	5	5
<u>Holding Cost</u>					
Product 1	1	1	2	2	2
Product 2	1	2	1	1	2

the production capacity constraints (12.31) are the coupling constraints. If there are no production capacity constraints, the problem reduces to  $N$  distinct lot sizing problems. Capacity is the only thing linking the products together.

In solving this problem with the column generation approach we effectively find the convex hull of constraint sets (12.32)-(12.35), that is, we do not solve the linear programming relaxation of the subproblems, we actually solve the subproblems as an integer program. Constraints (12.32)-(12.35) correspond to the set  $\Gamma$  in the development in the previous sections. Consider the data in Tables 12.1 and 12.2 for a two product, five time period problem.

Assume that the machine capacity is 200 units per period, regardless of which product is being produced on the machine. In order to start the process we need to generate some feasible schedules which generate columns for the restricted master.

The first solution to each of the products is the optimal solution assuming that there are no capacity constraints. The second solution for the first product is just the lot-for-lot solution, i.e. produce in every period exactly what is demanded for that period. Verify the costs associated with each of these schedules. Create a two convexity row restricted master where the  $z_{ij}$  variable corresponds to selecting schedule  $j$  for product  $i$ . The coefficients in the coupling constraints in the master are calculated in the usual way. Let  $A^1$  denote the constraint

**Table 12.2 Beginning Schedules**

Product 1						Cost
May	June	July	August	September		
60	240	0	200	120		5090
60	100	140	200	120		5250
Product 2						
May	June	July	August	September		Cost
100	0	140	0	80		1365

matrix of the coupling constraints. Then  $A^1 = [B^1, B^2, \dots, B^N]$  where each  $B^i$  matrix corresponds to the coefficients of product  $i$  in the coupling constraints. The only nonzeros of  $B^i$  correspond to the variables  $x_{it}$  where  $x_{it}$  has a coefficient of 1 in row  $t$  and a zero in the other rows. Therefore, the coefficient of  $z_{ij}$  in coupling constraint  $t$  of the master is simply the production quantity of schedule  $j$  in period  $t$ .

### First Restricted Master

$$\begin{array}{llllll}
 \min & 5090z_{11} & + & 5250z_{12} & + & 1365z_{21} \\
 & 60z_{11} & + & 60z_{12} & + & 100z_{21} \leq 200 & (\text{May}) \\
 & 240z_{11} & + & 100z_{12} & + & 0z_{21} \leq 200 & (\text{June}) \\
 & 0z_{11} & + & 140z_{12} & + & 140z_{21} \leq 200 & (\text{July}) \\
 & 200z_{11} & + & 200z_{12} & + & 0z_{21} \leq 200 & (\text{August}) \\
 & 120z_{11} & + & 120z_{12} & + & 80z_{21} \leq 200 & (\text{September}) \\
 & z_{11} & + & z_{12} & & = 1 \\
 & & & & z_{21} & = 1 \\
 & z_{ij} & \geq & 0 & & \\
 \end{array}$$

The solution to this problem is given below.

LP OPTIMUM FOUND AT STEP 3  
OBJECTIVE FUNCTION VALUE

1) 6500.71436

VARIABLE	VALUE	REDUCED COST
Z11	0.714286	0.000000
Z12	0.285714	0.000000
Z21	1.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	40.000000	0.000000
3)	0.000000	1.142859
4)	20.000000	0.000000
5)	0.000000	0.000000
6)	0.000000	0.000000
7)	0.000000	-5364.286133
8)	0.000000	-1365.000000

Why include column  $z_{12}$  in the first restricted master? Notice that the dual price for June is positive, i.e., this constraint is binding. Further observe that the optimal solutions of the individual subproblems require more resource than we have available in June. Thus, we will charge each division an extra \$1.142 per unit of product in June. This is reflected as an increase in the production marginal cost in the subproblem objective function. The subproblems are stated below. The objective function coefficients in the subproblems below are of the form  $(c^i - (-B^i)^\top u)$  (the minus sign on  $B^i$  since we must convert the capacity constraints into  $\geq$  for a minimization problem) where  $B^i$  represents the part of the coupling constraint matrix for product  $i$ . Each  $B^i$  has a column of 1's for each  $x_{it}$  variable, and zeros for all the other variables (only the  $x_{it}$  variables use capacity). The objective function coefficients are then unchanged for the  $y_{it}$  and  $I_{it}$  variables. For the  $x_{it}$  variables the cost is increased by  $u_t$  which can be thought of as a transfer price for using a scarce resource.

### Subproblem 1:

$$\begin{aligned} \min & (7 + u_1)x_{11} + (7 + u_2)x_{12} + (8 + u_3)x_{13} + (7 + u_4)x_{14} + (7 + u_5)x_{15} \\ & + I_{11} + I_{12} + 2I_{13} + 2I_{14} + 2I_{15} + 150y_{11} + 140y_{12} + 160y_{13} + 160y_{14} + 160y_{15} - u_{10} \end{aligned}$$

$$\begin{aligned} \text{s.t.} \quad x_{11} - I_{11} &= 60 \\ I_{11} + x_{12} - I_{12} &= 100 \\ I_{12} + x_{13} - I_{13} &= 140 \\ I_{13} + x_{14} - I_{14} &= 200 \\ I_{14} + x_{15} - I_{15} &= 120 \\ x_{11} - 620y_{11} &\leq 0 \end{aligned}$$

$$\begin{aligned}
 x_{12} - 620y_{12} &\leq 0 \\
 x_{13} - 620y_{13} &\leq 0 \\
 x_{14} - 620y_{14} &\leq 0 \\
 x_{15} - 620y_{15} &\leq 0 \\
 I_{1j}, \quad x_{1j} \geq 0, \quad y_{1j} \in \{0, 1\}
 \end{aligned}$$

**Subproblem 2:**

$$\begin{aligned}
 \min (2 + u_1)x_{21} + (4 + u_2)x_{22} + (4 + u_3)x_{23} + (5 + u_4)x_{24} + (5 + u_5)x_{25} \\
 + I_{21} + 2I_{22} + I_{23} + I_{24} + 2I_{25} + 30y_{21} + 30y_{22} + 40y_{23} + 50y_{24} + 35y_{25} - u_{20}
 \end{aligned}$$

$$\begin{aligned}
 x_{21} - I_{21} &= 40 \\
 I_{21} + x_{22} - I_{22} &= 60 \\
 I_{22} + x_{23} - I_{23} &= 100 \\
 I_{23} + x_{24} - I_{24} &= 40 \\
 I_{24} + x_{25} - I_{25} &= 80 \\
 x_{21} - 320y_{21} &\leq 0 \\
 x_{22} - 320y_{22} &\leq 0 \\
 x_{23} - 320y_{23} &\leq 0 \\
 x_{24} - 320y_{24} &\leq 0 \\
 x_{25} - 320y_{25} &\leq 0
 \end{aligned}$$

$$I_{2j}, \quad x_{2j} \geq 0, \quad y_{2j} \in \{0, 1\}$$

In formulating the subproblems the “Big M” values are not the smallest possible. However, the subproblems are not solved with an integer programming code. They are solved with dynamic programming (or the famous Wagner-Whitin algorithm [446]) or as shortest route problems. The columns generated using dynamic programming or a shortest route algorithm are not the same columns that would be generated by solving the linear programming relaxation of the subproblem. The dynamic programming and shortest route algorithms explicitly treat the fixed costs.

After solving the initial restricted master the subproblem objective functions have  $u_2 = 1.142$  the other  $u_i = 0$ ,  $u_{10} = 5364.29$ , and  $u_{20} = 1365$ . Now generate a new column for the master program by resolving subproblem 1 with the new objective function. (Why not add another column by resolving subproblem 2 with its new objective function?) The new subproblem 1 schedule is

May	June	July	August	September	Cost
300	0	0	200	120	5190

Since  $u_{10} = 5364.29$  a schedule with cost 5190 corresponds to a negative reduced cost of  $-174.29 = 5190 - 5364.29$ . Therefore, add a new column to the master program corresponding to  $z_{13}$ .

### Second Restricted Master

$$\begin{array}{rcccccccl}
 \min & 5090z_{11} & + & 5250z_{12} & + & 5190z_{13} & + & 1365z_{21} & \\
 & 60z_{11} & + & 60z_{12} & + & 300z_{13} & + & 100z_{21} & \leq 200 \\
 & 240z_{11} & + & 100z_{12} & + & 0z_{13} & + & 0z_{21} & \leq 200 \\
 & 0z_{11} & + & 140z_{12} & + & 0z_{13} & + & 140z_{21} & \leq 200 \\
 & 200z_{11} & + & 200z_{12} & + & 200z_{13} & + & 0z_{21} & \leq 200 \\
 & 120z_{11} & + & 120z_{12} & + & 120z_{13} & + & 80z_{21} & \leq 200 \\
 & z_{11} & + & z_{12} & + & z_{13} & & & = 1 \\
 & & & & & & & z_{21} & = 1 \\
 & z_{ij} & \geq & & 0 & & & & \\
 \end{array}$$

LP OPTIMUM FOUND AT STEP 3  
OBJECTIVE FUNCTION VALUE

1) 6471.66670

VARIABLE	VALUE	REDUCED COST
Z11	0.833333	0.000000
Z12	0.000000	0.000000
Z13	0.166667	0.000000
Z21	1.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	0.726190
3)	0.000000	1.142857
4)	60.000000	0.000000
5)	0.000000	0.000000
6)	0.000000	0.000000
7)	0.000000	-5407.857100
8)	0.000000	-1437.619000

May and June are now binding so we must charge for resource usage in these months. The new dual variables are  $u_1 = \$0.726190$ ,  $u_2 = \$1.142857$ ,  $u_3 =$

$u_4 = u_5 = 0$ ,  $u_{10} = 5407.86$  and  $u_{20} = 1437.62$ . Resolving the subproblems gives the following schedules.

<u>Product 1</u>					Cost
May	June	July	August	September	
160	0	140	200	120	\$ 5326.16

<u>Product 2</u>					Cost
May	June	July	August	September	
100	0	140	0	80	\$ 1437.60

### Third Restricted Master

$$\begin{aligned}
 \min \quad & 5090z_{11} + 5250z_{12} + 5190z_{13} + 5210z_{14} + 1365z_{21} \\
 & 60z_{11} + 60z_{12} + 300z_{13} + 160z_{14} + 100z_{21} \leq 200 \\
 & 240z_{11} + 100z_{12} + 0z_{13} + 0z_{14} + 0z_{21} \leq 200 \\
 & 0z_{11} + 140z_{12} + 0z_{13} + 140z_{14} + 140z_{21} \leq 200 \\
 & 200z_{11} + 200z_{12} + 200z_{13} + 200z_{14} + 0z_{21} \leq 200 \\
 & 120z_{11} + 120z_{12} + 120z_{13} + 120z_{14} + 80z_{21} \leq 200 \\
 & z_{11} + z_{12} + z_{13} + z_{14} = 1 \\
 & z_{21} = 1 \\
 z_{ij} & \geq 0
 \end{aligned}$$

1. Why wasn't a column added for the subproblem 2 solution?
2. Why is the objective function coefficient of  $z_{14}$  \$5210 instead of \$5326.16?

LP OPTIMUM FOUND AT STEP 5  
OBJECTIVE FUNCTION VALUE

1) 6471.66670

VARIABLE	VALUE	REDUCED COST
Z11	0.833333	0.000000
Z12	0.000000	81.666687
Z13	0.166667	0.000000
Z14	0.000000	0.000000

Z21	1.000000	0.000000
ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	0.142857
3)	0.000000	0.559524
4)	60.000000	0.000000
5)	0.000000	0.000000
6)	0.000000	0.000000
7)	0.000000	-5232.857100
8)	0.000000	-1379.285700
NO. ITERATIONS=	3	

The example problem is small and for pedagogical purposes only. In practice, such problems may have hundreds of thousands of columns since the number of extreme points and rays required to generate  $\Gamma$  is huge. These problems are seldom, if ever, solved to optimality. What we can do is establish a lower and upper bound on the optimal solution value of  $L(u)$  using Proposition 11.5 and terminate when the upper and lower bound are within an acceptable tolerance.

**Example 12.21 (Example 12.20)** Let  $z_{UB}$  denote the solution value of the optimal linear programming solution to the most recent master program. Since this is a feasible solution to the problem, and adding columns (unlike adding constraints) can only help,  $z_{UB}$  must be an upper bound on the optimal solution value of the Lagrangian dual problem. Let  $z_0(SP_i(\bar{u}, \bar{u}_{i0}))$  denote the optimal solution value to subproblem  $i$  for a dual solution  $\bar{u}, \bar{u}_{i0}$ . Direct application of Proposition 11.5 implies that a lower bound on the optimal solution of the full master is

$$z_0(SP_1(\bar{u}, \bar{u}_{10})) + z_0(SP_2(\bar{u}, \bar{u}_{20})) + \bar{u}_{10} + \bar{u}_{20} - \bar{u}^\top g.$$

These upper and lower bounds for the first three iterations are given in Table 12.3.

To see the lower bound for iteration 1 more explicitly, recall that the optimal solution values of subproblems 1 and 2 were, -174.29 and 0, respectively. The only nonzero dual variable on the coupling constraints was  $u_2 = 1.142$  and for the two convexity rows  $u_{10} = 5364.29$  and  $u_{20} = 1365$ . Then from Proposition 11.5 the lower bound is

$$6326.60 = -174.29 + 0 + 5364.29 + 1365.00 - (1.142)(200).$$

Calculate the lower bound for iteration 3.

**Table 12.3** Generating Upper and Lower Bound for Lot Sizing

	<i>Upper Bound</i>	<i>Lower Bound</i>
<i>Iteration 1</i>	6500.71	6326.60
<i>Iteration 2</i>	6471.67	6390.00
<i>Iteration 3</i>	6471.67	?

Finally, it is important to realize that the linear programming relaxation of the master problem is not the optimal solution to the lot sizing problem if there are fractional  $z_{ij}$  in the solution. This is because fractional variables in the master program correspond to fractional  $y_{it}$  variables. Refer back to the solution to the first restricted master and observe that for the first product  $z_{11} = 0.71$  and  $z_{12} = 0.29$ . In schedule one there is no production in July so  $y_{13} = 0$  in this schedule. However, in the second schedule there is a production of 140 units in July which means  $y_{13} = 1$ . Thus, the value of  $y_{13}$  in the second schedule is 0.29. It is possible to put a bound on the number of fractional solutions in a master problem.

Assume a production schedule is required for  $N$  products over  $T$  time periods. The resulting master program has  $T + N$  constraints. Of these constraints,  $N$  are *GUB* constraints. Proposition 12.22 is due to Manne [308].

**Proposition 12.22 (Manne)** *If  $N > T$ , then any basic feasible solution (i.e., an extreme point solution) of the restricted master has the property that at most  $T$  of the products have a fractional schedule.*

**Proof:** Any basic solution can have at most  $N + T$  variables in at a positive level. Each of the  $N$  *GUB* constraints requires at least one positive variable to satisfy it. The remaining  $T$  basic variables are then allocated to at most  $T$  different *GUB* rows leaving at least  $(N - T)$  of these with exactly one positive variable and this positive variable is one.  $\square$

In many practical problems, for example scheduling several hundred products over a twelve month horizon,  $N \gg T$ , leaving few fractional variables. We could then obtain a *heuristic integer solution* by solving an integer program over the basic variables. It is important to realize that even if the optimal solution to the master is integer, it is not necessarily an optimal solution to the lot sizing problem. Why is this so?

The treatment of setup costs in this example has not been particularly realistic. If a machine is setup (say with a particular configuration of dies) to produce a product in period  $t$ , and the same product is produced in period  $(t + 1)$ , then it might be reasonable to assume the setup cost in period  $(t + 1)$  is zero. We may wish to charge a setup cost if and only if the machine has to switch from production of one product to another. See Lasdon, [277] pp. 222-224 for a description of how setup costs were treated within a column generation framework for a real-world problem.

The dual of column generation is called *tangential approximation*. It can also be used to find the optimal value of the Lagrangian dual. See Geoffrion [172].

### 12.5.3 Method 3: Dual Ascent

For  $\bar{u} \geq 0$ , define

$$\partial L(\bar{u}) := \{\gamma \mid L(u) \leq L(\bar{u}) + (u - \bar{u})^T \gamma, \text{ for all } u \geq 0\}. \quad (12.36)$$

The set  $\partial L(\bar{u})$  defined in (12.36) is the *subdifferential* of  $L(u)$  at  $\bar{u}$ . It is the set of subgradients of the dual function  $L(u)$  at  $\bar{u}$ . It is left as an exercise to prove that the subdifferential is a closed convex set.

An important result (see Shapiro [405] or Rockafellar [382]) is that the directional derivative of  $L(u)$  at any point  $\bar{u}$ , in any direction  $d$ , exists and is given by

$$\begin{aligned} \nabla L(\bar{u}, d) = & \min_{\gamma \in \partial L(\bar{u})} d^T \gamma \\ \text{s.t.} & \gamma \in \partial L(\bar{u}). \end{aligned}$$

Given this result, one might use a steepest ascent approach to optimize  $L(u)$ . That is, solve the problem

$$\max_d \left[ \min_{\gamma \in \partial L(\bar{u})} d^T \gamma \right]$$

and use the resulting  $d$  as an ascent direction. Unfortunately, getting a handle on  $\partial L(\bar{u})$  is not easy and computational efforts in this direction (no pun intended) have been disappointing. See Shapiro [405]. Several researchers have, however, developed some very problem specific dual ascent algorithms. These methods tend to be heuristic in nature (convergence to an optimum not guaranteed) and very fast. Erlenkotter [143] and Bilde and Krarup [51] have applied heuristic procedures for finding optimal dual solutions to the simple plant location problem. Fisher, Jaikumar, and Van Wassenhove [150] developed a dual multiplier heuristic procedure for the generalized assignment problem.

## 12.6 COMPUTATIONAL ISSUES

It is crucial that the reader realize that if this material is to be useful, one must judiciously partition (*MIP*) into the coupling constraints  $Ax \geq b$  and the special structure constraints  $\Gamma$  so that for each  $u$  the Lagrangian problem  $L(u)$  is considerably easier to solve than the original primal problem. Suppose (*MIP*) had the following structure.

$$\begin{array}{lllllll} \min & c^1 x^1 & + & c^2 x^2 & + & \dots & + & c^m x^m \\ \text{s.t.} & A_1 x^1 & + & A_2 x^2 & + & \dots & + & A_m x^m & \geq & b \\ & B_1 x^1 & & & & & & & \geq & d^1 \\ & B_2 x^2 & & & & & & & \geq & d^2 \\ & & & & & \ddots & & & & \vdots \\ & & & & & & & & & \geq & d^m \\ & x^i \geq 0, x_j^i \in \mathcal{Z}, j \in I_i & & & & & & & & & \end{array}$$

This formulation is a block angular mixed integer linear program. The linear programming relaxation of this problem was studied in Chapter 11. By applying Lagrangian relaxation with respect to the coupling constraints we can solve  $m$  subproblems of the form

$$\begin{array}{ll} \min & (c^i - (A_i)^T u)^T x^i \\ (L_i(u)) \quad \text{s.t.} & B_i x^i \geq b^i \\ & x^i \geq 0 \quad x_j^i \in \mathcal{Z}, j \in I_i. \end{array}$$

These subproblems are used in Dantzig-Wolfe decomposition to price out the columns. The pricing operation involves solving  $m$  independent problems. We can expect the amount of effort required to solve an integer program to increase exponentially with the number of variables so we expect it to be much easier to solve  $m$  problems of form  $(L_i(u))$  rather than one problem involving all of the variables. Further computational enhancements are possible if the  $B^i$  have special structure.

**Example 12.23 (Example 12.19 Continued – Generalized Assignment Problem)** In the case of the generalized assignment problem when the assignment constraints  $\sum_{j=1}^m x_{ij} = 1$  are treated as the coupling constraints the Lagrangian problem decomposes into  $m$  binary knapsack algorithms for which specialized algorithms are available. The integrality property does not hold for this Lagrangian relaxation.

**Example 12.24 (Example 12.20 Continued – Dynamic Lot Sizing)** In the case of the multiproduct dynamic lot sizing problem when the capacity constraints  $\sum_{i=1}^N x_{it} \leq g_t$  are treated as the coupling constraints, the Lagrangian problem decomposes into  $N$  uncapacitated lot sizing problems for which specialized algorithms are available. The integrality property does not hold for this Lagrangian relaxation.

**Example 12.25 (Special Ordered Sets)** Special order sets of variables were studied in Chapter 9. If

$$B_i x^i = \{x^i \mid x_{i1} + x_{i2} + \dots + x_{in_i} = 1, x_{ij} \in \{0,1\}\}$$

then the solution to  $(L_i(u))$  is quite trivial. For  $i = 1, \dots, m$ , find the minimum component in each vector  $(c^i - (A_i)^\top u)$  and set the corresponding variable to one, all others go to zero. In the generalized assignment problem when the server capacity constraints  $\sum_{i=1}^n a_{ij} x_{ij} \leq b_j$  are dualized the remaining assignment constraints are special ordered sets of variables. The Lagrangian subproblem is now much easier to solve than when dualizing the assignment constraints since pegging variables is much easier than solving a knapsack problem. However, in this case the integrality property holds and the bounds are not as strong. Recall from Example 12.19 that the linear programming relaxation value was 6.42. By Corollary 12.9 the optimal value of the Lagrangian dual when the capacity constraints are relaxed is then 6.42. If the assignment constraints are dualized, the optimal value of the Lagrangian dual is 12.5.

**Example 12.26 (Simple Plant Location)** This problem was studied extensively in Chapter 10 on Benders' decomposition. Take the Lagrangian relaxation with respect to the demand constraints. Then there is a subproblem for each possible plant location. The subproblems have the following structure.

$$\begin{aligned} L_i(u) \quad & \min \sum_{j=1}^m (c_{ij} - u_j) x_{ij} + f_i y_i \\ \text{s.t.} \quad & x_{ij} \leq y_i, \quad i = 1, \dots, n, j = 1, \dots, m \\ & x_{ij} \geq 0, \quad i = 1, \dots, n, j = 1, \dots, m \\ & y_i \in \{0, 1\} \quad i = 1, \dots, n \end{aligned}$$

Once again, an optimal solution to  $(L_i(u))$  is easily found. If  $f_i + \sum_{j=1}^m \min\{0, (c_{ij} - u_j)\} < 0$ , then  $y_i = 1$ ,  $x_{ij} = 1$  if  $(c_{ij} - u_j) \leq 0$ ; else set all  $y_i$  and  $x_{ij} = 0$ . The integrality problem holds for this Lagrangian function.

How to partition the constraints into the sets  $Ax \geq b$  and  $x \in \Gamma$  is critical for the success of any implementation. Unfortunately, this is more of an art than science, and only comes with practice and experience. We now examine some important properties of our Lagrangian relaxation.

### Summary and Comments

**Selecting a Relaxation:** Problems with special structure may admit multiple possibilities for which constraints to dualize. There are several important considerations in selecting which constraints to dualize. Important issues in selecting the appropriate relaxation include: 1) choosing a relaxation which is significantly easier to solve than the original problem, 2) choosing a relaxation which is as tight as possible for bound generation, and 3) choosing a relaxation with a tendency to yield near feasible primal solutions. Unless the optimality conditions are satisfied the optimal dual solution does not provide a provably optimal primal solution. Rather than selecting a single relaxation, some researchers use a technique called *Lagrangian decomposition* which effectively dualizes more than one set of constraints. See, for example, Guignard and Kim [216].

**Branching:** Branching using Lagrangian relaxation is not as straightforward as it is when using linear programming relaxations. When using the linear programming relaxation a fractional variable is used to generate two new candidate problems. However, it is not always necessary to create two new candidate problems when branching. Examples where more than two new nodes are created are 3-machine flowshop (Ignall and Schrage [240]) and the generalized assignment problem. In the generalized assignment problem one can branch on the  $i$ th “task” constraint  $\sum_{j=1}^m x_{ij} = 1$  by creating  $m$  nodes with  $x_{ij} = 1$  at the  $j$ th node. This has the effect of reducing the size of the problem by one task.

**Generating Feasible Solutions:** How this is done is very problem specific. Obviously we would like to be able to generate feasible primal solutions easily so as to find tight upper bounds. Previous examples illustrate that when optimizing the Lagrangian dual there are often alternative optima to the problem  $L(u) = \min\{b^\top u + (c - A^\top u)^\top x \mid x \in \Gamma\}$ . A wise strategy for finding feasible solutions is to search among the alternative optima in  $\Gamma(u)$  and see if any of the alternative optima for the given  $u$  also satisfy the coupling  $Ax \geq b$  constraints.

**Generating the Multipliers:** There are four methods: 1) subgradient optimization, 2) column generation, 3) direct dual ascent and 4) heuristic methods.

In previous sections we have given references for “success stories” with each of methods except for direct dual ascent. The success of any of these methods depends very much on fine tuning the algorithm for the specific problem structure.

**Termination Criteria:** Here are two possible schemes:

*Method 1:* Use the result of Everett in Proposition 12.14 and terminate if there is an  $\bar{x} \in \Gamma$  and  $\bar{u}$  which satisfy complementary slackness,  $\bar{x}$  minimizes  $L(x, \bar{u})$  over  $\Gamma$ , and  $A\bar{x}$  is sufficiently close to  $b$ . If there are alternative  $x \in \Gamma$  which minimize  $L(x, \bar{u})$ , search among these solutions so that  $A\bar{x}$  is as close to  $b$  as possible.

*Method 2:* Use branch and bound with upper bounds (assume a minimization problem) generated by finding primal feasible solutions and lower bounds generated using Lagrangian relaxation. Terminate when the difference between the upper and lower bounds is within an acceptable tolerance.

## 12.7 A DECOMPOSITION ALGORITHM FOR INTEGER PROGRAMMING

The results in this section are taken from, “A Decomposition Algorithm for Integer Programming” by Sweeney and Murphy [418]. The methods of this section apply to problem *(BIP)* which is problem *(MIP)* with all variables binary. In many instances the Lagrangian dual provides very tight bounds on problem *(BIP)*. Thus, a good strategy might be to search among all optimal and/or near optimal solutions to a problem relaxation for an optimal or near optimal primal solution to this problem.

**Proposition 12.27** *Let  $\bar{x}$  be a feasible solution to *(BIP)* and  $c^\top \bar{x}$  its value. Given  $u \geq 0$ , let  $x^k$  denote the  $k$ th best solution to  $L(u) = \min\{b^\top u + (c - A^\top u)^\top x \mid x \in \Gamma\}$ . If  $c^\top x^k + u^\top (b - Ax^k) > c^\top \bar{x}$ , then the optimal solution  $x^*$ , must satisfy  $x^* \in \{x^1, x^2, \dots, x^{k-1}\}$ .*

**Corollary 12.28** *If the solutions in  $\{x^1, x^2, \dots, x^{k-1}\}$  which are feasible in *(BIP)* are re-ranked with  $u = 0$ , then the lowest ranking solution is optimal for *(BIP)*.*

Corollary 12.28 provides a method for finding an optimal solution to  $(BIP)$  if the conditions in Proposition 12.27 are satisfied. These two results suggest the following algorithm.

**Algorithm 12.29 (Ranking Algorithm)**

**Step 1:** Select a value of  $u \geq 0$ . (Perhaps the optimal linear programming or Lagrangian dual variables.)

**Step 2:** Rank order solutions to the Lagrangian problem until the  $k$ th solution satisfies the hypothesis of Proposition 12.27.

**Step 3:** Eliminate any of the  $k - 1$  solutions that are not feasible for  $(BIP)$ .

**Step 4:** Apply Corollary 12.28. Re-rank the remaining solution with respect to  $u = 0$  (i.e., the objective function for  $(BIP)$ ). Select as optimal the lowest ranking solution in this list.

The computational feasibility of this algorithm is dependent upon the size of  $k$  and the ease with which solutions can be ranked. Note that this procedure generates its own “incumbents.” Hence we could begin by choosing a  $u$  as in Step 1 above and then keep track of the best feasible solution to use in the termination criterion given by Proposition 12.27.

In general, this algorithm is not computationally tractable. At best we would have to generate all solutions within  $\epsilon$  of the optimum to  $L(u)$  in order to satisfy the hypothesis of Proposition 12.27. The minimum value of  $\epsilon$  is given by the size of the Lagrangian duality gap. There could be numerous solutions “inside” this gap. Consider again the generalized assignment problem.

**Example 12.30 (Example 12.19 Continued)**

$$\min \quad 9x_{11} + 2x_{12} + x_{21} + 2x_{22} + 3x_{31} + 8x_{32} \quad (12.37)$$

$$\text{s.t.} \quad x_{11} + x_{12} = 1 \quad (12.38)$$

$$x_{21} + x_{22} = 1 \quad (12.39)$$

$$x_{31} + x_{32} = 1 \quad (12.40)$$

$$6x_{11} + 7x_{21} + 9x_{31} \leq 13 \quad (12.41)$$

$$8x_{12} + 5x_{22} + 6x_{32} \leq 11 \quad (12.42)$$

$$x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in \{0, 1\} \quad (12.43)$$

Again dualize the assignment constraints and take advantage of the block angular structure that the generalized assignment problem exhibits.

Subproblem 1:

$$\begin{aligned} \min & (9 - u_1)x_{11} + (1 - u_2)x_{21} + (3 - u_3)x_{31} \\ \text{s.t.} & 6x_{11} + 7x_{21} + 9x_{31} \leq 13 \\ & x_{11}, x_{21}, x_{31} \in \{0, 1\} \end{aligned}$$

Subproblem 2:

$$\begin{aligned} \min & (2 - u_1)x_{12} + (2 - u_2)x_{22} + (8 - u_3)x_{32} \\ \text{s.t.} & 8x_{12} + 5x_{22} + 6x_{32} \leq 11 \\ & x_{12}, x_{22}, x_{32} \in \{0, 1\} \end{aligned}$$

For a given  $u$ , the optimal value of  $L(u)$  is the sum of the optimal values of the two subproblems plus  $u_1 + u_2 + u_3$ . For a given  $u$ , there are 25 possible solutions to  $L(u)$ . Why? In Table 12.4 are the rankings for the 25 possible dual solutions for three different dual vectors. The dual variables used are  $u = (0, 0, 0)$ ,  $u = (2, 2, 8)$  (the optimal linear programming dual variables) and  $u = (9, 7.5, 9.5)$  (the optimal Lagrangian dual variables).

Apply the algorithm for  $u = (9, 7.5, 9.5)$ . At worst, the 12th best solution generated is feasible ( $x_{11} = 1$ ,  $x_{21} = 1$ , and  $x_{32} = 1$ ) and gives an incumbent value  $c^\top \bar{x} = 18$ . Continuing the ranking of solutions the 13th best solution gives a value of 19 for the Lagrangian dual. Therefore, by Proposition 12.27 the optimal primal solution is contained among the 12 best solutions. Since  $X_{11} = 1$ ,  $X_{21} = 1$ , and  $X_{32} = 1$  is the only primal feasible solution in this set it must be optimal. If the optimal linear programming dual variables are used, how many solutions have to be generated in order to prove optimality?

At first inspection the ranking algorithm does not seem practical since it may require generating many ranked solutions to  $L(u)$ . However, if the problem  $L(u)$  separates into blocks, and all the variables are integer, one can use this rank ordering process to develop a master program analogous to the master program for Dantzig-Wolfe decomposition. Also, an exponential number of rank order solutions are generated from a polynomial number of solutions to each block. Consider the block angular binary integer program.

**Table 12.4** Solution Ranking For Example 12.30

	$x_{11}$	$x_{21}$	$x_{31}$	$x_{12}$	$x_{22}$	$x_{32}$	Feasible	$L(x, u)$		
							$u = (0, 0, 0)$	$u = (2, 2, 8)$	$u = (9, 7.5, 9.5)$	
$x^1$	0	0	0	0	0	0	No	0	12	26
$x^2$	0	0	0	1	0	0	No	2	12	19
$x^3$	0	0	0	0	1	0	No	2	12	20.5
$x^4$	0	0	0	0	0	1	No	8	12	24.5
$x^5$	0	0	0	0	1	1	No	10	12	19
$x^6$	1	0	0	0	0	0	No	9	19	26
$x^7$	1	0	0	1	0	0	No	11	19	19
$x^8$	1	0	0	0	1	0	No	11	19	20.5
$x^9$	1	0	0	0	0	1	No	17	19	24.5
$x^{10}$	1	0	0	0	1	1	Yes	19	19	19
$x^{11}$	0	1	0	0	0	0	No	1	11	19.5
$x^{12}$	0	1	0	1	0	0	No	3	11	12.5
$x^{13}$	0	1	0	0	1	0	No	3	11	14
$x^{14}$	0	1	0	0	0	1	No	9	11	18
$x^{15}$	0	1	0	0	1	1	No	11	11	12.5
$x^{16}$	0	0	1	0	0	0	No	3	7	19.5
$x^{17}$	0	0	1	1	0	0	No	5	7	12.5
$x^{18}$	0	0	1	0	1	0	No	5	7	14
$x^{19}$	0	0	1	0	0	1	No	11	7	18
$x^{20}$	0	0	1	0	1	1	No	13	7	12.5
$x^{21}$	1	1	0	0	0	0	No	10	18	19.5
$x^{22}$	1	1	0	1	0	0	No	12	18	12.5
$x^{23}$	1	1	0	0	1	0	No	12	18	14
$x^{24}$	1	1	0	0	0	1	Yes	18	18	18
$x^{25}$	1	1	0	0	1	1	No	20	18	12.5

$$\begin{array}{llllll}
 \min & c^1 x^1 & + & c^2 x^2 & + & \dots + & c^m x^m \\
 \text{s.t.} & A_1 x^1 & + & A_2 x^2 & + & \dots + & A_m x^m \geq b \\
 & B_1 x^1 & & & & & \geq d^1 \\
 & & B_2 x^2 & & & & \geq d^2 \\
 & & & \ddots & & & \vdots \\
 & & & & B_m x^m & \geq & d^m \\
 & & x^i \geq 0, & & x_j^i \in \{0, 1\} & &
 \end{array}$$

Applying Lagrangian relaxation to the coupling constraints yields

$$\begin{array}{ll}
 \min & (c^i - (A_i)^T u)^T x^i \\
 (L_i(u)) \quad \text{s.t.} & B_i x^i \geq b^i \\
 & x_j^i \in \{0, 1\}
 \end{array}$$

Let  $x^{ij}$  be the  $j$ th best solution to  $(L_i(u))$ . Define the following master program  $(MP)$ .

$$\begin{array}{ll}
 \min & \sum_{j=1}^{k_1} f_{1j} z_{1j} + \sum_{j=1}^{k_2} f_{2j} z_{2j} + \dots + \sum_{j=1}^{k_m} f_{mj} z_{mj} \\
 (MP) \quad \text{s.t.} & \sum_{j=1}^{k_1} p^{1j} z_{1j} + \sum_{j=1}^{k_2} p^{2j} z_{2j} + \dots + \sum_{j=1}^{k_m} p^{mj} z_{mj} \geq b_0 \\
 & \sum_{j=1}^{k_i} z_{ij} = 1, \quad i = 1, \dots, m \\
 & z_{ij} \in \{0, 1\}, \quad i = 1, \dots, m, \quad j = 1, 2, \dots, k_i
 \end{array}$$

where,  $f_{ij} = c^i x^{ij}$ ,  $p^{ij} = A^i x^{ij}$ . Here  $f_{ij}$  represents the value of the  $j$ th best solution to subproblem  $i$  and  $p^{ij}$  represents the amount of shared resources consumed or provided by the  $j$ th best solution to subproblem  $i$ .

Define  $\Delta := z_{UB} - L(u)$  ( $z_{UB}$  is the best known solution value of  $(MP)$ ) and

$$\delta_i := ((c^i - (A^i)^T u)^T x^{ik_i} - (c^i - (A^i)^T u)^T x^{i1}), \quad i = 1, 2, \dots, m.$$

**Proposition 12.31** *If  $\delta_i > \Delta$  for  $i = 1, 2, \dots, m$  then any optimal solution to  $(MP)$  is an optimal solution to  $(BIP)$ .*

When  $m = 1$ , Proposition 12.27 implies Proposition 12.31. The significance of the case of  $m > 1$  is that when  $\min_{1 \leq i \leq m} \{\delta_i\} > \Delta$ , the set of  $\Delta$ -optimal solutions to  $L(u)$  is contained in the Cartesian product of

$$\{x^{11}, \dots, x^{1k_1}\} \times \{x^{21}, \dots, x^{2k_2}\} \times \cdots \times \{x^{m(1)}, \dots, x^{mk_m}\}.$$

This represents  $\prod_{i=1}^m k_i$  solutions! However, all solutions within  $\Delta$  of the optimal solution to  $L(u)$  are generated by the fewer  $\delta_i$ -optimal subproblem solutions. We can implicitly generate all solutions within  $\Delta$  of the optimum to  $L(u)$  in a master program consisting of only  $\sum_{i=1}^m k_i$  columns! See also Exercise 11.6.

Limited computational evidence indicates that this procedure can be effective. Sweeney and Murphy [418] report results on a block angular problem that originally had 2486 0-1 variables and 1400 constraints. The problem consisted of 167 coupling constraints with the other constraints partitioned into 10 subproblems. They generated 20 rank order solutions to each  $L_i(u)$  with  $u$  equal to the optimal linear programming dual variables on the coupling constraints. This gave a master program with 200 0/1 variables and 167 coupling constraints and 10 special ordered set rows. They were able to find and prove optimality of a solution in approximately 1/2 hour on an old IBM 370. Piper and Zoltners [371] describe a linear programming based branch-and-bound algorithm for rank ordering solutions to binary linear programs.

## 12.8 CONCLUSION

Until the early 1970s linear programming based branch-and-bound was the most widely used method for solving integer programming problems. Then, in 1970 and 1971, Held and Karp published [224, 225] two papers using a Lagrangian duality approach for solving the traveling salesman problem. Based upon the Held and Karp work, the earlier work of Everett [145], and their own work, Geoffrion [174], Fisher [148], and others popularized the Lagrangian dual as another means of generating bounds on the optimal integer linear programming solution value. Although, the idea of “dualizing” a constraint with a Lagrange multiplier and Lagrangian function goes back to Lagrange [275] in 1797, results such as those in Proposition 12.8 and the concept of using a Lagrangian dual function to generate bounds within an enumeration scheme were new to the mathematical programming community. This was an enormous leap forward in the field of computational integer programming. Lagrangian techniques are still very important today. With so many major advancements

in linear programming technology, column generation techniques developed in Subsection 12.5.2 combined with a branch-and-bound algorithm are becoming very popular for solving the convex hull relaxation of problem (*MIP*). See Section 16.4 on branch-and-price in Chapter 16.

## 12.9 EXERCISES

- 12.1 Prove Proposition 12.12.
- 12.2 Take the simple plant location model in Section 11.3 of Chapter 11 and perform three iterations of subgradient optimization. In this example you should dualize the demand constraints.
- 12.3 Assume the point  $\bar{u} \geq 0$  is *not* an optimal dual solution of  $L(u)$ . Then every subgradient gives a direction of ascent. Prove, or give a counter example.
- 12.4 The Sender company is in the process of planning for new production facilities and in order to develop a more efficient distribution system design. At present they have one plant in St. Louis with a capacity of 30,000 units. Because of increased demand, management is considering four potential new plant sites: Detroit, Denver, Toledo, and Kansas City. The transportation tableau below summarizes the projected plant capacities, the cost per unit of shipping from each plant to each customer zone, and the demand forecasts over a one-year planning horizon. The fixed costs of constructing the new plants are: Detroit \$175,000, Toledo \$300,000, Denver \$375,000 and Kansas City \$500,000.
- Formulate this as a mixed 0/1 linear program.
  - Dualize the demand constraint and optimize the resulting Lagrangian dual.

Plant	Zone 1	Zone 2	Zone 3	Supply
Detroit	5	2	3	10,000
Toledo	4	3	4	20,000
Denver	9	7	5	30,000
Kansas	10	4	2	40,000
St. Louis	8	4	3	30,000
Demand	30,000	20,000	20,000	

- 12.5 Find the optimal value of the Lagrangian dual for the multiproduct dynamic lot sizing problem in Example 12.20 using subgradient optimization.
- 12.6 Prove Proposition 12.27.
- 12.7 Apply the Ranking Algorithm to Example 12.30 with  $u = (10, 8, 10)$ .
- 12.8 Prove Proposition 12.31.
- 12.9 Formulate the model given by (12.30)-(12.35) using the data in Example 12.20 and solve using a linear programming based branch and bound code. Make one run where  $M_{it} = 10,000$ , a second run with  $M_{it} = \sum_{l=1}^T d_{il}$  and a third run where  $M_{it} = \min\{g_t, \sum_{l=t}^T d_{il}\}$ . Compare the three runs in terms of total pivots, total branches, and the gap between the initial LP and the optimal integer programming solution.
- 12.10 Calculate the production quantities, inventory levels, and value of the  $y_{it}$  variables associated with the solution to the third restricted master program.
- 12.11 Verify that
- $$z_0(SP_1(\bar{u}, u_{10})) + z_0(SP_2(\bar{u}, u_{20})) + u_{10} + u_{20} - u^1 b^1$$
- is a valid lower bound for the lot sizing master program.
- 12.12 Calculate the lower bound for the third iteration of the lot sizing example.
- 12.13 Continue the column generation algorithm for the lot sizing example until an optimal solution is found to the master program.
- 12.14 Prove Proposition 12.22.
- 12.15 Prove Corollary 12.6.
- 12.16 If the linear program (*LP*) is infeasible, then the dual problem is either infeasible or unbounded. If the mixed integer program (*MIP*) is infeasible, is it true that the Lagrangian dual formed by dualizing the  $Ax \geq b$  is either infeasible or unbounded?
- 12.17 Prove that the subdifferential of a function defined in (12.36) is a closed convex set.

## SPARSE METHODS

### 13.1 INTRODUCTION

In order to solve linear programs of realistic size it is necessary to take advantage of the sparsity of the constraint matrix. In this chapter we study several matrix factorizations which are used in simplex and barrier algorithms. In Section 13.2 we introduce *LU factorization*, which is used in Gaussian elimination and the simplex algorithm. In Section 13.3 we describe methods for updating the *LU* factorization of a basis during simplex pivots. For sparse problems, the *LU* factorization and update of a basis matrix generally require fewer nonzero elements than the product form of the inverse. This leads to faster FTRAN and BTRAN operations which are required for finding the updated pivot column and the pivot row of the basis inverse. In Sections 13.4 and 13.5 we present algorithms for *Cholesky factorization*. The Cholesky factorization is the most time consuming step of barrier algorithms. In implementing sparse algorithms for matrix factorization it is important to work with only the nonzero matrix elements. In Section 13.6 we describe data structures for storing sparse matrices. Several compiler and implementation issues related to sparse matrices are discussed in Section 13.7. In Section 13.8 we briefly discuss the current state of barrier and simplex codes. Concluding remarks are in Section 13.9. Exercises are provided in Section 13.10.

### 13.2 LU DECOMPOSITION

In Chapter 2 we introduced projection, or Gaussian elimination, as an algorithm for solving the system  $Ax = b$ . In this section we show how Gaussian elimination is actually implemented in practice for solving large linear systems. Recall, that

if we project out variable  $x_1$  from the system of equations  $\sum_{j=1}^n a_{ij}x_j = b_i$ ,  $i = 1, \dots, m$ , using the first equation (assume  $a_{11} \neq 0$ ) the projected system in the variable space  $x_2, \dots, x_n$  is

$$\sum_{j=2}^n (a_{2j} - (a_{11}/a_{11})a_{1j})x_j = b_i - a_{11}(b_1/a_{11}), \quad i = 2, \dots, m.$$

Then in  $x_1, \dots, x_n$  space, a system equivalent (a system with exactly the same solution set) to  $Ax = b$  is

$$\begin{aligned} \sum_{j=1}^n a_{1j}x_j &= b_1 \\ \sum_{j=2}^n (a_{2j} - (a_{11}/a_{11})a_{1j})x_j &= b_i - a_{11}(b_1/a_{11}), \quad i = 2, \dots, m. \end{aligned}$$

If  $A$  is an  $m \times n$  matrix of rank  $r \geq 1$ , repeating the projection process yields a sequence of matrices  $A_1, A_2, \dots, A_{\hat{r}}$  where  $\hat{r} = \max\{r, 2\}$  and the elements of  $A_k$  for  $k = 1, \dots, \hat{r} - 1$  are:

$$a_{ij}^{(k+1)} = a_{ij}^k - (a_{ik}^k/a_{kk}^k)a_{kj}^k, \quad i, j > k \quad (13.1)$$

$$a_{ij}^{(k+1)} = a_{ij}^k, \quad i \leq k \quad (13.2)$$

$$a_{ij}^{(k+1)} = 0, \quad i > k \text{ and } j \leq k \quad (13.3)$$

$$b_i^{(k+1)} = b_i^k - (a_{ik}^k/a_{kk}^k)b_k^k, \quad i > k \quad (13.4)$$

$$b_i^{(k+1)} = b_i^k, \quad k \leq i, \quad (13.5)$$

and

$$A_{\hat{r}} = \begin{bmatrix} B & C \\ 0 & 0 \end{bmatrix}$$

where  $B$  is an  $r \times r$  upper triangular matrix with nonzero elements on the diagonal. Each system  $A_k x = b^k$  is equivalent to the original system. The process in (13.1)-(13.5) is initialized by  $A_1 = A$ ,  $b^1 = b$ .

In (13.1) and (13.4) element  $a_{kk}^k$  is called the *pivot element*. In these recursive equations we assume that the pivot element is nonzero. If this is not the case, then either a row or column is permuted to make  $a_{kk}^k$  nonzero. First we show that if no row or column permutations are required then  $A = LA_{\hat{r}}$  where  $L$  is a lower triangular  $m \times m$  matrix. When  $A$  is a square matrix,  $A_{\hat{r}}$  is an upper

triangular matrix  $U$  and  $A$  is written as  $A = LU$ . This is the *LU decomposition* of  $A$ . Define for  $i > k$ ,

$$l_{ik} = a_{ik}^k / a_{kk}^k. \quad (13.6)$$

Then pivot row  $k$  is multiplied by  $l_{ik}$  and subtracted from row  $i$ ,  $i > k$ . Define the  $m \times m$  matrix  $L_k$  by

$$L_k = \begin{bmatrix} 1 & & & \\ 0 & \ddots & & \\ \vdots & & 1 & \\ 0 & & l_{k+1,k} & \\ \vdots & & \vdots & \ddots \\ 0 & & l_{m,k} & & 1 \end{bmatrix}.$$

Then

$$L_k^{-1} = \begin{bmatrix} 1 & & & \\ 0 & \ddots & & \\ \vdots & & 1 & \\ 0 & & -l_{k+1,k} & \\ \vdots & & \vdots & \ddots \\ 0 & & -l_{m,k} & & 1 \end{bmatrix}$$

and (13.1)-(13.3) imply for  $k = 1, \dots, \hat{r} - 1$

$$A_{k+1} = L_k^{-1} A_k \quad (13.7)$$

and

$$A_{\hat{r}} = L_{(\hat{r}-1)}^{-1} L_{(\hat{r}-2)}^{-1} \cdots L_1^{-1} A \quad (13.8)$$

Then from (13.8)

$$A = L_1 L_2 \cdots L_{\hat{r}-1} A_{\hat{r}} = L A_{\hat{r}} \quad (13.9)$$

where

$$L = L_1 L_2 \cdots L_{\hat{r}-1} = \begin{bmatrix} 1 & & & \\ l_{21} & \ddots & & \\ \vdots & & 1 & \\ l_{\hat{r},1} & & l_{\hat{r},\hat{r}-1} & \\ \vdots & & \vdots & \ddots \\ l_{m1} & & l_{m,\hat{r}-1} & \cdots & 1 \end{bmatrix}$$

This discussion is easily modified to treat the case when rows or columns need to be permuted so that the pivot element is nonzero. Consider the following example.

**Example 13.1**

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & -1 & 1 \\ 2 & 2 & 6 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 4 & 0 & 4 & 1 & -1 & 1 \end{bmatrix}$$

Pivot on element (1,1). This gives:

$$A_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 1 \end{bmatrix} \quad \text{and} \quad L_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Since  $\bar{a}_{22}^2 = 0$  it is necessary to exchange row 2 with another row. This is accomplished by multiplying  $A_2$  by a permutation matrix. A permutation matrix is the identity matrix with rows or columns exchanged. Pre-multiply  $A_2$  with the permutation matrix defined by exchanging the second and third rows of a  $5 \times 5$  identity matrix. Call this matrix  $P_{23}$ . Then

$$\bar{A}_2 := P_{23}A_2 = P_{23}L_1^{-1}A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 1 \end{bmatrix}$$

Since the elements  $\bar{a}_{32}^2, \bar{a}_{42}^2$  and  $\bar{a}_{52}^2$  are zero no row operations are necessary using pivot element  $\bar{a}_{22}^2$ . Then  $L_2^{-1}$  is an identity matrix and  $A_3 = L_2^{-1}\bar{A}_2$ .

Because  $\bar{a}_{33}^2 = \bar{a}_{43}^2 = \bar{a}_{53}^2 = 0$  it is not possible to get a nonzero pivot element by a row exchange. It is possible, to get one by exchanging columns 3 and 4. This is done by post multiplying matrix  $A_3$  by the permutation matrix obtained by exchanging columns 3 and 4 of the  $5 \times 5$  identity matrix. Call this matrix

$Q_{34}$ . Then

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 \end{bmatrix} = L_2^{-1} P_{23} L_1^{-1} A Q_{34}$$

Now pivot on element (3, 3) and obtain the matrices

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad L_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Exchange columns 4 and 5 and get

$$\bar{A}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = L_3^{-1} L_2^{-1} P_{23} L_1^{-1} A Q_{34} Q_{45}$$

The following results are left as exercises.

**Lemma 13.2** If  $A$  is an  $m \times m$  nonsingular matrix then there exists an upper triangular matrix  $U$ , lower triangular matrices  $L_1, \dots, L_{m-1}$  with diagonal elements equal to 1.0 and permutation matrices  $P_k$  such that

$$U = L_{m-1}^{-1} P_{m-1} L_{m-2}^{-1} P_{m-2} \cdots L_1^{-1} P_1 A.$$

**Proposition 13.3** If  $A$  is an  $m \times m$  nonsingular matrix then there exists an upper triangular matrix  $U$ , lower triangular matrix  $L$  with diagonal elements equal to 1.0 and a permutation matrix  $P$  such that

$$LU = PA.$$

Assume that  $A$  is a nonsingular  $m \times m$  matrix. One method for solving the system  $Ax = b$  is to calculate  $A^{-1}$  which gives  $x = A^{-1}b$ . A better solution method is to use the decomposition of  $A$  given by  $PA = LU$ . Write the system  $Ax = b$  as

$$PAx = LUx = Pb. \quad (13.10)$$

This system is now solved in a two step process. First solve the system

$$Ly = Pb. \quad (13.11)$$

Since  $L$  is a lower triangular matrix with diagonal elements of 1.0, the inverse  $L^{-1}$  is calculated from  $L$  by simply negating the elements below the diagonal (actually,  $L^{-1}$  is generated when generating the  $LU$  decomposition of  $A$ ). Given  $L^{-1}$ , the solution of (13.11) is  $\bar{y} = L^{-1}Pb$ . Given  $\bar{y}$ , solve the system

$$Ux = \bar{y} \quad (13.12)$$

for  $\bar{x}$ . Because  $U$  is an upper triangular matrix, system (13.12) is solved easily by *back substitution*. Denote element  $(i, j)$  of  $U$  by  $u_{ij}$ .

#### Algorithm 13.4 (Back Substitution)

```

for i = m, ..., 2
     $x_i \leftarrow \bar{y}_i / u_{mm}$ 
    for j = 1, ..., i - 1
         $\bar{y}_j \leftarrow \bar{y}_j - x_i u_{ji}$ 
    end
end
 $x_1 \leftarrow \bar{y}_1 / u_{11}$ 
```

It is left as Exercise 13.8 to explain why it is better solve system (13.12) through back substitution, rather than finding  $U^{-1}$  and calculating  $x = U^{-1}\bar{y}$ . In Section 13.3 we show that the FTRAN and BTRAN operations at each pivot of revised simplex are better carried out using  $LU$  decomposition rather than the product form of the inverse.

### 13.2.1 Partial Pivoting

In practice Gaussian elimination is not done with infinite precision. Regardless of the precision used (single or double) there are a fixed number of bits allocated for each number and rounding errors occur. For example, even the number 0.1 cannot be represented exactly in binary in the computer. Rounding error can obviously have a very deleterious effect on solution quality. The row permutations used in doing the pivoting can have a significant effect on solution accuracy. Partial pivoting exploits this fact. Recall Example 6.13 used to illustrate the minimum ratio test. The reason for using the Harris ratio test was to

allow for larger pivot elements in order to improve the numerical accuracy of the simplex method. We continue with that example and show the problem of selecting small pivot elements.

**Example 13.5 (Example 6.13 continued)**

$$0.1x_1 + 100x_2 = .0009 \quad (13.13)$$

$$100x_1 + 100x_2 = 1 \quad (13.14)$$

Assume that our “computer” has three digits of accuracy, that is, it stores only the three most significant digits after rounding. Pivoting on  $x_1$  in (13.13) gives

$$\begin{aligned} 0.1x_1 + 100x_2 &= .0009 \\ -100000x_2 &= .1 \end{aligned}$$

Back solving yields the three digit solution  $\hat{x}_1 = 1 \times 10^{-3}$ ,  $\hat{x}_2 = -1 \times 10^{-6}$

Now do this computation in infinite precision. Pivot on  $x_1$  and get:

$$\begin{aligned} -99900x_2 &= .1 \\ x_2 &= -1.001 \times 10^{-6} \end{aligned}$$

Substitute back into (13.13) to get the true solution which rounded to three digits is  $\bar{x}_1 = 1. \times 10^{-2}$ ,  $\bar{x}_2 = -1.00 \times 10^{-6}$ . The solution value for  $x_1$  calculated in finite precision differs from the true value by a factor of 10.

Now exchange rows (13.13) and (13.14) and again pivot on  $x_1$  using three digit precision in each step. This gives

$$\begin{aligned} 100x_1 + 100x_2 &= 1 \\ 100x_2 &= -.0001 \end{aligned}$$

and the 3 digit solution is now  $\hat{x}_1 = 1. \times 10^{-2}$  and  $\hat{x}_2 = -1. \times 10^{-6}$  which agrees exactly with the infinite precision solution rounded to three digits.

This example suggests a strategy due to Wilkinson [449] called *partial pivoting*. With partial pivoting at pivot  $k$  the rows  $i \geq k$  are reordered so that

$$|a_{kk}^{(k)}| \geq |a_{ik}^{(k)}|, \quad i > k.$$

### 13.2.2 Nonzero Fill In

The row permutation used can also affect the number of nonzeros in the updated matrix. Consider the matrix below where an \* indicates a nonzero matrix element.

$$A = \begin{bmatrix} * & * & * & \cdots & * & * \\ * & * & & & & \\ * & & * & & & \\ \vdots & & & \ddots & & \\ * & & & & * & \\ * & & & & & * \end{bmatrix}$$

If a pivot is done on element  $a_{11}$  then matrix  $A_2$  will be 100 percent dense. The changing of zero elements to nonzero elements as a result of a pivot is called *fill-in*. Fill-in is bad for two reasons. First, when decomposing  $A$  into  $L$  and  $U$ , fill-in means that more memory is required to store  $L$  and  $U$ . If there were no fill-in, (for example,  $A$  is upper triangular) then  $L$  is the identity matrix and the number of nonzeros in  $U$  is equal to the number of nonzeros in  $A$ . This is significant when  $A$  is sparse. Second, the sparser  $L$  and  $U$  are, the less computation required when solving the systems  $Ly = Pb$  and  $Ux = y$ . Therefore, one would like to have a pivot strategy which results in very little fill-in. One such strategy is that of Markowitz [309] (also a winner of the Nobel prize for his work in portfolio theory). The basic idea of Markowitz is to observe that if a pivot is done on element  $a_{kk}^k$  and there are  $\eta_k$  nonzero elements in column  $k$  in rows  $i \geq k$  and  $s_k$  nonzero elements in row  $k$  for columns  $j \geq k$ , then the potential fill-in is  $(\eta_k - 1) \times (s_k - 1)$ . The Markowitz criterion is to select the nonzero pivot element which minimizes this product.

Tomlin [428] suggests using a compromise of the partial pivoting and minimum nonzero fill in strategies. Given column  $k$ , the pivot row selected is

$$i^* = \operatorname{argmin}\{s_i \mid |a_{i+k}^{(k)}| \geq u \max_{i \geq k} |a_{ik}^{(k)}|\}$$

where  $u$  is a relative tolerance. Tomlin gives computational results with  $u$  between .00001 and 1. A  $u = 1$  corresponds to the partial pivoting strategy and a  $u = 0$  corresponds to the Markowitz rule.

### 13.3 SPARSE LU UPDATE

Two key operations in the revised simplex method are updating the pivot column  $a^q$  (Step 3) and finding the pivot row  $i^*$  of the basis inverse  $A_B^{-1}$  (Step 5). Using the product form of the inverse, the basis inverse is written as (ignoring permutations)  $A_B^{-1} = E_m E_{m-1} \cdots E_2 E_1$  where the  $E_i$  are the eta matrices described in Chapter 5. Using the product form of the inverse, the column update is the FTRAN operation

$$\bar{a}^q = E_m E_{m-1} \cdots E_2 E_1 a^q$$

and row  $i^*$  of  $A_B^{-1}$  is found using the BTRAN operation

$$(z^{i^*})^T = (e^{i^*})^T E_m E_{m-1} \cdots E_2 E_1.$$

A crucial factor affecting the efficiency of the BTRAN and FTRAN is the sparsity of the eta vectors. The fewer nonzero elements in the eta file, the faster the BTRAN and FTRAN operations.

In the previous section, we saw a nonsingular  $m \times m$  matrix  $A_B$  can be factored, ignoring row permutations, as  $A_B = LU$  where  $L$  is lower triangular and  $U$  is upper triangular. The  $LU$  decomposition actually produces  $L^{-1}$  as the product of the lower triangular matrices  $L_{m-1}^{-1} L_{m-2}^{-1} \cdots L_1^{-1}$ . If  $A_B$  is factored this way, the column update operation  $\bar{a}^q = A_B^{-1} a^q$  is equivalent to  $LU \bar{a}^q = a^q$  which is done by calculating  $\hat{a}^q$  using the FTRAN

$$\hat{a}^q = L_{m-1}^{-1} L_{m-2}^{-1} \cdots L_1^{-1} a^q$$

and then solving  $U \bar{a}^q = \hat{a}^q$  for  $\bar{a}^q$  by back substitution. Be careful of the notation,  $\bar{a}^q = A_B^{-1} a^q$  and  $\hat{a}^q = L^{-1} a^q$  where  $A_B = LU$ .

Similarly, the BTRAN operation  $(z^{i^*})^T = (e^{i^*})^T A_B^{-1}$  is equivalent to  $(z^{i^*})^T LU = (e^{i^*})^T$  which is done by first solving  $y^T U = (e^{i^*})^T$  for  $y^T$  by *forward substitution* and then calculating  $(z^{i^*})^T L = y^T$  for  $(z^{i^*})^T$  by the BTRAN operation

$$(z^{i^*})^T = y^T L_{m-1}^{-1} L_{m-2}^{-1} \cdots L_1^{-1}.$$

The basis matrix changes with each simplex pivot and is periodically reinverted. Markowitz [309] was the first to use the  $LU$  factorization each time the basis matrix was reinverted. However, Markowitz used the product form of the inverse during pivoting after refactoring the basis. Another possibility is to recalculate the  $LU$  factorization after each simplex pivot. This is what most modern commercial codes do. The potentially big advantage of  $LU$  factorization over the product form of the inverse is that the number of nonzero

elements in  $L$  and  $U$  may be much less than in the eta vectors used in the product form of the inverse. This allows for faster pricing and dual variable calculation. If the product form of the inverse is used, at each pivot the new eta vector contains one nonzero element for each nonzero element of  $\bar{a}^q = A_B^{-1}a^q$ . The  $LU$  update, if done properly, may require adding far fewer nonzero elements than the number of nonzero elements in  $\bar{a}^q$ . This is particularly true on large sparse problems with special structure. The rest of this section is devoted to efficient strategies for updating the  $LU$  factorization of the basis matrix at each simplex pivot.

Assume that at the current simplex iteration column  $q$  is selected to pivot into the basis in row  $i^*$ . Thus, the new basis is  $\hat{B} = B \setminus \{B_{i^*}\} \cup \{q\}$  and the new basis matrix is

$$A_{\hat{B}} = A_B + (a^q - A_B e^{i^*})(e^{i^*})^T.$$

At this point the reader may wish to refer back to the Sherman-Morrison-Woodbury rank one update formula given in Lemma 5.8 in Chapter 5. Multiplying by  $L^{-1}$  where  $A_B = LU$  gives

$$L^{-1}A_{\hat{B}} = U + (L^{-1}a^q - U e^{i^*})(e^{i^*})^T$$

so  $L^{-1}A_{\hat{B}}$  is equal to  $U$  except for column  $i^*$  which has been replaced by  $\hat{a}^q = L^{-1}a^q$ . This new column is called a *spike* and is illustrated in (13.15).

$$L^{-1}A_{\hat{B}} = \left[ \begin{array}{cccccccccccc} * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \end{array} \right] \quad (13.15)$$

The presence of the spike column can cause considerable fill-in inverting  $L^{-1}A_{\hat{B}}$  because the nonzero elements below the diagonal must be eliminated. Bartels and Golub [40] were the first to propose a method for updating and maintaining the sparsity of the  $LU$  factorization at each simplex pivot. In order to reduce fill-in as much as possible, the Bartels-Golub update permutes  $L^{-1}A_{\hat{B}}$  as follows.

1. Exchange column  $m$  with column the column  $\hat{a}^q$  which is in position  $i^*$ .
2. Move columns  $i^* + 1$  through  $m$  one position to the left.

This is accomplished by post-multiplying  $L^{-1}A_{\hat{B}}$  by the permutation matrix  $Q$  which has  $e^i$ , for columns  $i = 1, \dots, i^* - 1$ ,  $e^{i+1}$  for columns  $i = i^*, \dots, m - 1$  and  $e^i$  for column  $m$ . The resulting matrix is an upper *Hessenberg* matrix and is illustrated below in (13.16).

$$L^{-1}A_{\hat{B}}Q = \begin{bmatrix} * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \end{bmatrix} \quad (13.16)$$

Bartels and Golub bring  $L^{-1}A_{\hat{B}}Q$  into upper triangular form by reducing the subdiagonal elements  $u_{i^*+1,i^*+1}, \dots, u_{mm}$ . If there are no pivot row exchanges then the element below the diagonal in column  $j$  is eliminated by pre-multiplying  $A_{\hat{B}}Q$  by the matrix  $L_j$  defined in (13.17).

$$L_j = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\frac{u_{j+1,j+1}}{\hat{a}_{j,j+1}} & 1 & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} \quad (13.17)$$

We write  $\hat{a}_{j,j+1}$  instead of  $u_{j,j+1}$  to indicate that the element in this position may have been modified when another subdiagonal element was eliminated. The eta vector corresponding to matrix  $L_j$  has only two nonzero elements.

In order to maintain numerical stability, Bartels and Golub employ the partial pivoting idea and use a row interchange so as to pivot on the largest element of

$u_{j+1,j+1}, \hat{u}_{j,j+1}$ . If there is an exchange of pivot rows,  $L^{-1}A_{\tilde{B}}Q$  is pre-multiplied by

$$L_j = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -\frac{\hat{u}_{j,j+1}}{u_{j+1,j+1}} & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} \quad (13.18)$$

Unfortunately, although the  $L_j$  are very sparse with only one off-diagonal element, the Bartels Golub procedure may lead to considerable fill-in for the part of the upper Hessenberg matrix in columns  $i^* + 1$  to  $m$  and rows  $i^* + 1$  to  $m$ . Thus, the  $\hat{L}\hat{U}$  decomposition of  $A_{\tilde{B}}$  has a sparse  $\hat{L}$  but dense  $\hat{U}$ . When this algorithm was first proposed, this was potentially a significant problem as it made it difficult to store  $U$  in core. It also made updating  $U$  stored in the packed form discussed later difficult. An alternate  $LU$  update was proposed by Forrest and Tomlin [155]. The key idea behind the Forrest-Tomlin update is to permute the rows of  $L^{-1}A_{\tilde{B}}Q$  by  $Q^T$ . Doing so has the effect of replacing the column spike in  $L^{-1}A_{\tilde{B}}$  with a row spike row in the last row<sup>1</sup>. With this method, there is no fill-in for the matrix  $Q^T L^{-1} A_{\tilde{B}} Q$ . In (13.19)

$$Q^T L^{-1} A_{\tilde{B}} Q = \begin{bmatrix} * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \end{bmatrix} \quad (13.19)$$

we show the effect of premultiplying the upper Hessenberg matrix  $L^{-1}A_{\tilde{B}}Q$  by  $Q^T$ . We can then premultiply this matrix by a lower triangular eta matrix  $E^{-1}$

<sup>1</sup>Actually, Forrest and Tomlin first eliminate the elements of  $L^{-1}A_{\tilde{B}}Q$ , in row  $i^*$ , for columns  $i^*$  to  $m-1$ . This leads to a row permuted upper triangular matrix and pre-multiplying this matrix by the permutation matrix  $Q^T$  results in an upper triangular matrix.

so that  $E^{-1}Q^T L^{-1} A_{\bar{B}} Q$  is an upper triangular matrix with the same density as  $U$  except that column  $i^*$  of  $U$  is replaced by column  $\hat{a}^q$  in  $\hat{U}$ . The matrix  $E^{-1}$  is calculated as follows. Define

$$E := I_m + e^m \eta^T = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & 1 & \\ \eta_{i^*} & \cdots & \eta_{m-1} & 1 \end{bmatrix} \quad (13.20)$$

where

$$\eta^T = [ 0 \ \cdots \ 0 \ \eta_{i^*} \ \cdots \ \eta_{m-1} \ 0 ] \quad (13.21)$$

is an  $m$  vector. Component  $i^*$  of  $\eta$  is  $\eta_{i^*}$  and  $\eta^T$  is the solution to the system

$$\eta^T Q^T U Q = \bar{u}^T \quad (13.22)$$

where

$$\bar{u}^T = [ 0 \ \cdots \ 0 \ u_{i^*, i^*+1} \ \cdots \ u_{i^*, m} \ 0 ].$$

In the  $m$  vector  $\bar{u}$ , component  $i^*$  is  $u_{i^*, i^*+1}$ . If  $E$  is given as in (13.20), then  $E^{-1} = I_m - e^m \eta^T$  and

$$\begin{aligned} E^{-1} Q^T U Q &= (I_m - e^m \eta^T) Q^T U Q \\ &= Q^T U Q - e^m \bar{u}^T Q^T U^{-1} Q Q^T U Q \quad \text{from (13.22)} \\ &= Q^T U Q - e^m \bar{u}^T. \end{aligned}$$

Thus,  $E^{-1} Q^T U Q$  is equal  $Q^T U Q$  except for the elements in row  $m$ , columns  $i^*, \dots, m-1$  which are zero in  $E^{-1} Q^T U Q$ . Therefore,  $E^{-1} Q^T U Q$  is upper triangular with nonzero diagonal elements. The matrix  $Q^T U Q$  is identical to the matrix  $Q^T L^{-1} A_{\bar{B}} Q$  except for column  $m$ . This implies

$$\hat{U} := E^{-1} Q^T L^{-1} A_{\bar{B}} Q \quad (13.23)$$

is an upper triangular matrix and, except for perhaps the last column, contains no more non-zeros than in the original  $U$ . Therefore, the maximum number of new non-zeros that must be stored with the pivot on variable  $q$  are the elements  $\eta_{i^*}, \dots, \eta_{m-1}$  in the new eta matrix  $E$  plus perhaps, some new nonzero elements in column  $m$  of  $\hat{U}$ . With the product form of the inverse each eta vector can contain as many as  $m-1$  new non-zero elements.

If

$$\hat{L}^{-1} := E^{-1}Q^T L^{-1}, \quad (13.24)$$

then  $\hat{L}\hat{U} = A_{\hat{B}}Q$  and we have an  $LU$  decomposition of the new (permuted) basis matrix.

A further improvement in the Forrest-Tomlin update method is due to Saunders [397]. With the Saunders update, the last nonzero element  $\hat{a}_{sq}$  in the spike column  $\hat{a}^q = L^{-1}a^q$  is used in determining the permutation of  $L^{-1}A_{\hat{B}}$ . Rather than permuting the last column of  $L^{-1}A_{\hat{B}}$  with the spike column, the spike column is exchanged with column  $s$  and columns  $i^* + 1, \dots, s$  are moved one position to the left to form an upper Hessenberg matrix. The submatrix consisting of the rows and columns  $i^*$  through  $s$  is called a *bump*. After creating the upper Hessenberg matrix, the rows are permuted so there is a single spike row. This spike row is now row  $s$ , rather than row  $m$ . Compare (13.25) below to (13.19).

$$\left[ \begin{array}{cccccccccccc} * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * & * & * & * & * \end{array} \right] \quad (13.25)$$

Assume the last nonzero of column  $L^{-1}A_{\hat{B}}$  is in row  $s$ . Modify the definitions of  $E$  and  $\eta^T$  to

$$E = I_m + e^s \eta^T \quad (13.26)$$

where

$$\eta^T = [ 0 \quad \cdots \quad 0 \quad \eta_{i^*} \quad \cdots \quad \eta_{s-1} \quad 0 \quad \cdots \quad 0 ] \quad (13.27)$$

where  $\eta$  is an  $m$  vector,  $\eta_{i^*}$  is component  $i^*$  of  $\eta$  and  $\eta^T$  is the solution to the system

$$\eta^T Q^T U Q = \bar{u}^T \quad (13.28)$$

where

$$\bar{u}^T = [ 0 \ \cdots \ 0 \ u_{i^*, i^*+1} \ \cdots \ u_{i^*, s} \ 0 \ \cdots \ 0 ].$$

The maximum number of elements in the eta matrix  $E$  is now  $s - i^*$  compared to  $m - i^*$  with the Forrest-Tomlin method. Algorithm 13.6 below combines ideas of Bartels and Golub, Forrest and Tomlin and Saunders. See also Suhl and Suhl [415].

### Algorithm 13.6 (LU Update)

*Calculate the spike column  $L^{-1}a_q$  and the pivot row  $i^*$*

*Define the columns  $q^i$  of permutation matrix  $Q$  by:*

*Initialize  $q^i \leftarrow e^i$ ,  $i = 1, \dots, m$*

*for  $i = i^*, \dots, s - 1$*

*$q^i \leftarrow e^{i+1}$*

*end*

*$q^s \leftarrow e^{i^*}$*

*Calculate  $E$  from (13.26)*

*Replace  $U$  with  $\hat{U} \leftarrow E^{-1}Q^T L^{-1} A_B Q$  and  $L^{-1}$  with  $L^{-1} \leftarrow E^{-1}Q^T L^{-1}$ .*

Algorithm LU Update produces an upper triangular matrix  $\hat{U}$ . This LU update is then incorporated into the revised simplex method for pricing and column updating as follows.

### Algorithm 13.7 (Revised Simplex with LU Update)

**Step 1: (Initialization)** Given a basis matrix  $A_B$  such that  $A_B^{-1}b \geq 0$ , initialize the primal and dual solution

$$\bar{x}_B \leftarrow A_B^{-1}b, \bar{z} \leftarrow c_B^T A_B^{-1}b, \bar{u}^T \leftarrow c_B^T A_B^{-1}, \bar{w}_N^T \leftarrow c_N^T - \bar{u}^T A_N.$$

**Step 2: (Pivot Column Selection)** If  $\bar{w}_N \geq 0$ , stop, the current solution is optimal; else, select  $q \in N$  such that  $\bar{w}_q < 0$ .

**Step 3 (Update the Pivot Column)** Calculate the updated column  $\bar{a}^q$

$$\bar{a}^q \leftarrow A_B^{-1}a^q.$$

Do so by solving the system

$$A_B \bar{a}^q = LU \bar{a}^q = a^q.$$

This system is solved in two steps, do an FTRAN to calculate  $\hat{a} \leftarrow L^{-1}\bar{a}^q$  and then a back solve  $U\bar{a}^q = \hat{a}^q$  to find  $\bar{a}^q$ .

**Step 4: (Primal Minimum Ratio Test and Pivot Row Selection)** If  $\bar{a}_{iq} \leq 0$  for  $i = 1, \dots, m$  stop, the linear program is unbounded. Otherwise, perform the primal minimum ratio test

$$i^* \leftarrow \operatorname{argmin} \{\bar{x}_{B_i}/\bar{a}_{iq} \mid \bar{a}_{iq} > 0, i = 1, \dots, m\}$$

Update the primal and dual step lengths.

$$\alpha_P \leftarrow \bar{x}_{B_{i^*}}/\bar{a}_{i^*q}, \quad \alpha_D \leftarrow \bar{w}_q/\bar{a}_{i^*q}$$

**Step 5: (Row Update)** Find row  $i^*$  of  $A_B^{-1}$  by solving the system

$$(z^{i^*})^T A_B = (z^{i^*})^T LU = (e^{i^*})^T.$$

This system is solved in two steps, use forward substitution to calculate  $y^T U = (e^{i^*})^T$  and then a BTRAN to find  $(z^{i^*})^T = y^T L^{-1}$ .

**Step 6: (Update Primal and Dual Solution, Basis and Basis Inverse)**

$$\bar{x}_{B_i} \leftarrow \bar{x}_{B_i} - \alpha_P \bar{a}_{iq}, \quad i = 1, \dots, m, \quad \bar{x}_q \leftarrow \alpha_P, \quad \bar{z} \leftarrow \bar{z} + \alpha_P \bar{w}_q$$

$$\bar{u}^T \leftarrow \bar{u}^T + \alpha_D z^{i^*}, \quad \bar{w}_N^T \leftarrow \bar{w}_N^T - \alpha_D (z^{i^*})^T A_N, \quad \bar{w}_{B_{i^*}} \leftarrow -\alpha_D$$

Update the index of basic variables.

$$B \leftarrow (B \setminus \{B_{i^*}\}) \cup \{q\}, \quad N \leftarrow (N \setminus \{q\}) \cup \{B_{i^*}\}$$

and update the LU decomposition using Algorithm 13.6. Go to Step 2.

**Example 13.8 (Example 5.2 continued)** Recall the example linear program (in standard form) from Chapter 5.

$$\begin{array}{rccccccccc} z & & +5x_3 & -2x_4 & = & -1 \\ x_1 & & +x_3 & -x_4 & = & 1 \\ x_2 & & -x_3 & & = & 1 \\ x_5 & & +1.5x_3 & -.5x_4 & = & 7.5 \\ x_6 & & +2x_3 & -x_4 & = & 6 \end{array}$$

**Iteration 1:**

**Step 1 (Initialization)** The initial basis matrix is

$$A_B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}$$

corresponding to variables  $x_1, x_2, x_5$  and  $x_6$ . The first step in finding the LU decomposition of  $A_B$  is exchanging the first and second row in order to get a nonzero in element (1,1). This corresponds to pre-multiplying  $A_B$  by a permutation matrix  $P_{12}$  with the first and second rows exchanged.

$$P_{12}A_B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}$$

Pivot on element (1,1). This involves multiplying row one by (1/2) and adding the result to row three and then adding row one to row four. This corresponds to pre-multiplying  $P_{12}A_B$  by the matrix

$$E_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Then

$$E_1 P_{12} A_B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{3}{2} & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

Next pivot on (2,2). This corresponds to pre-multiplying by

$$E_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{3}{2} & 1 & 0 \\ 0 & -2 & 0 & 1 \end{bmatrix}$$

Then  $L^{-1} = E_2 E_1 P_{12}$  and

$$L^{-1} A_B = E_2 E_1 P_{12} A_B = U = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The eta matrices  $E_1, E_2$  and the permutation matrix  $P_{12}$  are now used for the revised simplex calculations.

For this basis, the initial values of the dual variables and reduced costs are

$$\bar{u}^T = [-5 \ 2 \ 0 \ 0], \quad \bar{w}_N^T = c_N^T - \bar{u}^T A_N = [-5 \ 2].$$

**Step 2: (Pivot Column Selection)** Variable  $x_3$  is the only variable with a negative reduced cost, select it to enter the basis.

**Step 3: (Update the Pivot Column)** The eta file is used to calculate the updated column  $\bar{a}^3$  by solving  $LU\bar{a}^3 = a^3$ . First calculate  $\hat{a}^3 = L^{-1}a^3 = E_2E_1P_{12}a^3$  using an FTRAN. Let  $\eta^i$  be the appropriate eta vector.

$$\begin{aligned}\hat{a}^3 &\leftarrow P_{12}a^3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \\ \hat{a}^3 &\leftarrow E_1\hat{a}^3 = \hat{a}^3 + (\eta^1 - e^1)(e^1)^T\hat{a}^3 = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \\ \hat{a}^3 &\leftarrow E_2\hat{a}^3 = \hat{a}^3 + (\eta^2 - e^2)(e^2)^T\hat{a}^3 = \begin{bmatrix} 0 \\ -1 \\ \frac{3}{2} \\ 2 \end{bmatrix}\end{aligned}$$

Finally, solve  $U\bar{a}^3 = \hat{a}^3$  and obtain solution  $\bar{a}^3$ .

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{a}_1^3 \\ \bar{a}_2^3 \\ \bar{a}_3^3 \\ \bar{a}_4^3 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ \frac{3}{2} \\ 2 \end{bmatrix}.$$

The solution is

$$\bar{a}^3 = \begin{bmatrix} 1 \\ -1 \\ \frac{3}{2} \\ 2 \end{bmatrix}$$

**Step 4: (Minimum RatioTest)** As seen previously,  $i^* = 1$  and variable  $x_1$  leaves the basis. At this iteration,  $\bar{a}_{13} = 1$ ,  $\bar{x}_1 = 1$  and  $\bar{w}_3 = -5$ . Then the primal and dual step lengths are  $\alpha_P = 1$  and  $\alpha_D = -5$ .

**Step 5: (Row Update)** Find row one of  $A_B^{-1}$ . Solve the system  $(z^1)^T LU = (e^1)^T$  for  $z^1$ . First solve  $y^T U = (e^1)^T$ .

$$\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The solution of this system is  $y_1 = 1$ ,  $y_2 = -1$ ,  $y_3 = 0$ ,  $y_4 = 0$ . Now solve  $(z^1)^T L = y^T$  with the BTRAN operation  $y^T L^{-1}$ .

$$(z^1)^T \leftarrow y^T E_2 = y^T + (y^T \eta^2 - y^T e^2)(e^2)^T = [ 1 \ -1 \ 0 \ 0 ]$$

$$(z^1)^T \leftarrow (z^1)^T E_1 = (z^1)^T + ((z^1)^T \eta^1 - (z^1)^T e^1)(e^1)^T = [ 1 \ -1 \ 0 \ 0 ]$$

$$(z^1)^T \leftarrow (z^1)^T P_{12} = [ 1 \ -1 \ 0 \ 0 ] \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [ -1 \ 1 \ 0 \ 0 ]$$

**Step 6: (Update Primal and Dual solutions, Basis and Basis Inverse)**

The new primal solution is

$$\begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_5 \\ \bar{x}_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 7.5 \\ 6 \end{bmatrix} - 1 \begin{bmatrix} 1 \\ -1 \\ 1.5 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 6 \\ 4 \end{bmatrix}$$

and  $\bar{x}_3 = 1$ .

The updated dual variables using (5.28)-(5.29) are

$$\begin{aligned} \hat{u}^T &= \bar{u}^T + \alpha_D(z^1)^T \\ &= [ -5 \ 2 \ 0 \ 0 ] - 5 [ -1 \ 1 \ 0 \ 0 ] = [ 0 \ -3 \ 0 \ 0 ] \end{aligned}$$

$$\begin{aligned} \hat{w}_{\hat{N}}^T &= \bar{w}_{\hat{N}}^T - \alpha_D(z^1)^T A_{\hat{N}} \\ &= [ 0 \ 2 ] + 5 [ -1 \ 1 \ 0 \ 0 ] \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = [ 5 \ -3 ] \end{aligned}$$

The new basis is  $\hat{B} = \{3, 2, 5, 6\}$ . Update the basis and LU factorization. The new basis is  $x_3, x_2, x_5$  and  $x_6$  so the new basis matrix is

$$A_{\hat{B}} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

This basis differs from the previous basis in the first column only. From the FTRAN in Step 3,  $\hat{a}^3 = L^{-1}a^3$  so replace the first column in  $U$  with  $\hat{a}^3$  and get

$$L^{-1}A_{\hat{B}} = E_2E_1P_{12}B = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ \frac{3}{2} & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix}.$$

The first column is the spike column and the last nonzero in this column is in the last row (row four). Following Bartels and Golub, and Forrest and Tomlin, make column 1 the last column and move columns 2 through 4 forward 1 column. That is,

$$L^{-1}A_{\hat{B}}Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & \frac{3}{2} \\ 0 & 0 & 1 & \frac{1}{2} \end{bmatrix}$$

where

$$Q = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Then

$$\hat{U} = Q^T L^{-1} A_{\hat{B}} Q = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & \frac{3}{2} \\ 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad (13.29)$$

Next zero out the elements in row  $m$  (four) and columns 1 through  $m-1$  (one through three).

$$E^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

This gives

$$\hat{U} = E^{-1}Q^T L^{-1} A_B Q = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & \frac{3}{2} \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this example  $\hat{a} = L^{-1}a^3$  has three nonzero elements and  $\bar{a} = A_B^{-1}a^3$  has four nonzero elements. Then the new eta vector using the product form of the inverse requires storing four nonzero elements. The  $LU$  update requires one new nonzero to be stored in  $E^{-1}$ . In the upper triangular matrix  $\hat{U}$ , has two more nonzero elements than  $U$ . Then  $LU$  decomposition for this pivot requires only three additional nonzero elements instead of four. The next example gives an even more dramatic illustration of how the  $LU$  basis update can preserve sparsity better than the product form of the inverse.

A further improvement on the Saunders method suggested by Reid [378] is to reduce the size of the bump matrix prior to performing the row operations to eliminate nonzero elements below the diagonal. This has the potential to reduce even further, the number of new nonzero elements. For example, if there are any singleton columns in the bump, the nonzero element in the singleton column can be moved into position  $(i^*, i^*)$  by row and column permutations thus reducing the size of the bump. Similarly, the nonzero element in any singleton row can be moved to the lower rightmost part of the bump through row and column permutations. A slight disadvantage of this is that the row and column permutations are not necessarily symmetric and two permutation matrices are required.

**Example 13.9** Consider the following basis matrix  $A_B$  and nonbasic column  $a^q$ .

$$A_B = \begin{bmatrix} 1 & -1 & & & & \\ & 1 & & & & \\ & & 1 & -1 & & \\ & & & 1 & -1 & \\ & & & & 1 & -1 \\ & & & & & 1 \end{bmatrix}, \quad a^q = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Then

$$\bar{a}^q = A_B^{-1}a^q = \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

If the product form of the inverse is used, then the new eta vector added when  $a^q$  is added to the basis will have five nonzero elements. However, since  $A_B$  is upper triangular, its LU decomposition is  $U = A_B$  and  $L = I$ . Then  $\hat{a}^q = L^{-1}a^q = Ia^q = a^q$ . Assume the first row is the pivot row. The updated basis matrix is

$$L^{-1}A_{\hat{B}} = A_{\hat{B}} = \begin{bmatrix} & & -1 & & & & \\ -1 & 1 & & & & & \\ & & 1 & -1 & & & \\ & & & 1 & -1 & & \\ 1 & & & & 1 & -1 & \\ & & & & & 1 & -1 \\ & & & & & & 1 \end{bmatrix}$$

The last nonzero element is in row five so the bump matrix consists of rows and columns one through five. First put the basis matrix into upper Hessenberg form.

$$A_{\hat{B}}Q = \begin{bmatrix} 1 & -1 & & & & & \\ & 1 & -1 & & & & -1 \\ & & 1 & -1 & & & \\ & & & 1 & -1 & & \\ & & & & 1 & -1 & \\ & & & & & 1 & -1 \\ & & & & & & 1 \end{bmatrix}$$

Next, perform the symmetric row permutations and transform the matrix so there is a spike row in row five.

$$Q^T A_{\hat{B}} Q = \begin{bmatrix} 1 & & & & -1 & & \\ & 1 & -1 & & & & \\ & & 1 & -1 & & & \\ & & & 1 & 1 & -1 & \\ -1 & & & & & 1 & -1 \\ & & & & & & 1 \end{bmatrix}$$

Apply the procedure of Reid and move singleton columns to the top of bump and singleton rows to the bottom of the bump. The bump matrix is in rows and columns one through five, so we drop the last two rows and columns. The first column of the bump is a singleton column with the nonzero in the first row so the bump matrix is actually in rows and columns two through five. In this four by four bump matrix the last row and column are singletons. Exchange the first and last column of the bump matrix.

$$\left[ \begin{array}{ccccc} 1 & & & -1 & \\ & 1 & -1 & & \\ & & 1 & -1 & \\ & & & 1 & 1 \\ -1 & & & & \end{array} \right] \rightarrow \left[ \begin{array}{cccc} 1 & -1 & -1 & 1 \\ & 1 & -1 & \\ 1 & & 1 & \\ & & & -1 \end{array} \right]$$

Now exchange the first and third rows of the bump matrix.

$$\left[ \begin{array}{ccccc} 1 & -1 & & & \\ & -1 & & 1 & \\ & & 1 & -1 & \\ 1 & & 1 & & \\ & & & -1 & \end{array} \right] \rightarrow \left[ \begin{array}{ccccc} 1 & -1 & & & \\ & 1 & & & \\ & & 1 & -1 & \\ & & & -1 & 1 \\ & & & & -1 \end{array} \right]$$

Now the bump matrix is in rows and columns three and four. The second bump row is a singleton row and the second bump column is a singleton column. Exchange the first and second columns in the bump matrix defined by rows and columns three and four.

$$\left[ \begin{array}{ccccc} 1 & -1 & & & \\ & 1 & & 1 & \\ & & 1 & -1 & \\ -1 & & & 1 & \\ & & & -1 & \end{array} \right] \rightarrow \left[ \begin{array}{ccccc} 1 & -1 & & & \\ & 1 & & 1 & \\ & & -1 & 1 & \\ & & & -1 & 1 \\ & & & & -1 \end{array} \right]$$

The new basis is in upper triangular form. No nonzero elements are added to the eta file for this pivot using the LU decomposition method. This example is a linear programming network flow problem. It is discussed in greater detail in Chapter 14. It is a good illustration of a problem structure for which the product form of the inverse may behave very poorly compared to the LU decomposition. For this problem structure, a cycle is created when nonbasic variable  $x_q$  is added to the basis. The updated column  $\bar{\alpha}^q$  has a nonzero element for each arc in the cycle. Again, see Chapter 14. This can cause a considerable accumulation of nonzero elements when using the product form of the inverse.

The programming aspects of the FTRAN and BTRAN are described in later in Section 13.7.

## 13.4 NUMERIC CHOLESKY FACTORIZATION

Like simplex based algorithms, barrier algorithms must also take advantage of matrix sparsity in order to solve large, realistic problems. All barrier algorithms require solving a linear system of the form  $ADA^T x = b$  where  $A$  has full row rank and  $D$  is a diagonal matrix with strictly positive diagonal elements. It is left as an exercise to show that this implies  $M = ADA^T$  is a positive definite matrix. An  $m \times m$  matrix  $M$  is positive definite if and only if  $x^T M x > 0$  for all nonzero  $x \in \mathbb{R}^m$ .

When using Gaussian elimination to solve the system  $Mx = b$ , the matrix  $M$  is factored as  $M = LU$  where  $L$  is lower triangular with diagonal elements of 1.0, and  $U$  is upper triangular. In this section, we show that if  $M$  is a symmetric positive definite matrix, then  $M$  can be factored as the product  $M = LL^T$  where  $L$  is a lower triangular matrix. This special structure factorization is called a *Cholesky factorization*. This factorization arose originally from solving a least squares application arising in triangulating the island of Crete and was developed by a Major André-Louis Cholesky in the French Geographic Service. For an interesting history of Cholesky factorization from its inception to the present see Saunders [396].

We motivate the Cholesky factorization by looking at an important application in portfolio optimization. Let  $r_i$  be the random variable corresponding to the return of stock  $i$  and let  $M$  be the variance-covariance matrix of two stocks.

$$M = \begin{bmatrix} u_{11} & u_{21} \\ u_{21} & u_{22} \end{bmatrix}.$$

In matrix  $M$ ,  $u_{ii}$  is the variance of stock  $i$  and  $u_{ij}$  is the covariance between stocks  $i$  and  $j$ ,  $i \neq j$ . See, for example, Markowitz [310] for further discussion on how this matrix is constructed from historical data. We wish to generate a set of scenarios or outcomes, say for a simulation experiment, which duplicate the variance-covariance matrix  $M$ . See, for example, Schrage [402] for applications of scenarios in portfolio optimization. Many simulation and spreadsheet packages allow one to generate independent pseudo random numbers for such simulations. In the real world, random variables tend to be correlated. An interesting question is, “can we generate a set of random variables having a specified correlation structure?” It turns out we can by taking appropriate linear combinations of independent random variables which have a covariance of 0.

If the single random variable  $r_1$  is a nonzero multiple of the random variable  $z_1$ , i.e.  $r_1 = l_{11}z_1$ , then

$$u_{11} = \text{Var}(r_1) = \text{Var}(l_{11}z_1) = l_{11}^2$$

which implies

$$l_{11} = \sqrt{u_{11}}. \quad (13.30)$$

Next, duplicate the variance-covariance matrix of a two stock portfolio by taking a linear combination of two independent random variables, each with variance 1.0. If the first random variable is defined by  $r_1 = l_{11}z_1$  where  $l_{11} = \sqrt{u_{11}}$ , then the second random variable is defined by  $r_2 = l_{21}z_1 + l_{22}z_2$ . Clearly,  $l_{21}$  cannot be zero when  $u_{21} \neq 0$  since  $z_1$  and  $z_2$  are independent. First, calculate the value of  $l_{21}$  so that  $r_1$  and  $r_2$  have covariance equal to  $u_{21}$ .

$$u_{21} = \text{covariance}(r_1, r_2) = \text{covariance}(l_{11}z_1, l_{21}z_1 + l_{22}z_2) \quad (13.31)$$

$$= \text{covariance}(l_{11}z_1, l_{21}z_1) + \text{covariance}(l_{11}z_1, l_{22}z_2) \quad (13.32)$$

$$= l_{11}l_{21} \quad (13.33)$$

Thus,  $l_{21} = u_{21}/l_{11} = u_{21}/\sqrt{u_{11}}$ . Next, calculate the multiplier  $l_{22}$  on  $z_2$  so that  $r_2$  has the appropriate variance  $u_{22}$ .

$$\begin{aligned} u_{22} &= \text{variance}(r_2) = \text{variance}(l_{21}z_1 + l_{22}z_2) \\ &= \text{variance}(l_{21}z_1) + \text{variance}(l_{22}z_2) \\ &= l_{21}^2 + l_{22}^2 \end{aligned}$$

Therefore,  $l_{22} = \sqrt{u_{22} - l_{21}^2} = \sqrt{u_{22} - (u_{21}^2/u_{11})}$ .

In general, if stock  $i$  is added to a portfolio with  $i - 1$  stocks, the covariance between  $i$ , and a stock  $j < i$  already in the portfolio, is

$$\begin{aligned} u_{ij} &= \text{covariance}(l_{i1}z_1 + \dots + l_{ii}z_i, l_{j1}z_1 + \dots + l_{jj}z_j) \\ &= l_{i1}l_{j1} + \dots + l_{ij}l_{jj} \end{aligned}$$

Thus, finding the covariance of stock  $i \geq 2$  with stocks  $j = 1, \dots, i - 1$ , already in the portfolio, requires the solution of the triangular system

$$\left[ \begin{array}{cc} l_{11} & 0 \\ \vdots & \ddots \\ l_{i-1,1} & \cdots & l_{i-1,i-1} \end{array} \right] \left[ \begin{array}{c} l_{i1} \\ \vdots \\ l_{i,i-1} \end{array} \right] = \left[ \begin{array}{c} u_{i1} \\ \vdots \\ u_{i,i-1} \end{array} \right]. \quad (13.34)$$

The variance of stock  $i$  is

$$u_{ii} = \text{variance}(l_{i1}z_1 + \cdots + l_{ii}z_i) = l_{i1}^2 + \cdots + l_{ii}^2.$$

The values of  $l_{i1}, \dots, l_{i,i-1}$  are known from the solution of (13.34) so  $l_{ii}$  is given by

$$l_{ii} = \sqrt{u_{ii} - \sum_{j=1}^{i-1} l_{ij}^2}. \quad (13.35)$$

Combining (13.30) for  $i = 1$ , with (13.34) and (13.35) for  $i = 2, \dots, n$  gives a lower triangular matrix  $L = [l_{ij}]$  such that  $LL^T = M$ . The logic of (13.30)-(13.35) applies to any symmetric positive definite matrix.

**Proposition 13.10** *If  $M$  is an  $n \times n$  positive definite matrix, then there exists a unique  $n \times n$  lower triangular matrix  $L$  with positive diagonal elements such that  $M = LL^T$ .*

The matrix  $L$  is known as the *Cholesky factorization* of  $M$ . The details of the proof of Proposition 13.10 are left as an exercise. The proof is an inductive one. If an  $(n-1) \times (n-1)$  symmetric positive definite matrix  $C$  can be factored as  $L_C L_C^T$ , then the  $n \times n$  symmetric positive definite matrix  $M$  can be written as

$$M = \begin{bmatrix} C & u \\ u^T & s \end{bmatrix}$$

where  $C = L_C L_C^T$ ,  $u$  is an  $n-1$  vector and  $s$  is a scalar. Why does  $M$  positive definite imply  $C$  is positive definite? Next, solve the system  $L_C w = u$  for  $w$ . This is what we do in (13.34). Then

$$\begin{bmatrix} L_C & 0 \\ w^T & (s - w^T w)^{1/2} \end{bmatrix} \begin{bmatrix} L_C^T & w \\ 0 & (s - w^T w)^{1/2} \end{bmatrix} = \begin{bmatrix} C & u \\ u^T & s \end{bmatrix}$$

Why is  $(s - w^T w)$  is positive?

The major step in calculating the Cholesky factorization is (13.34). The best way to implement (13.34) depends greatly on the computer architecture (e.g. memory access patterns, vectorization, parallelism, etc.) and programming language (e.g. FORTRAN or C). For an excellent discussion of how these issues impact Gaussian elimination see Dongarra, Gustavson, and Karp [129]. One implementation of (13.34) is given below in Algorithm 13.11. Following the terminology of Dongarra, Gustavson, and Karp we call this an  $ijk$  implementation due to the loop ordering.

**Algorithm 13.11 (Row- $ijk$  Cholesky Factorization)**

```

 $l_{11} \leftarrow \sqrt{u_{11}}$ 
for  $i = 2, \dots, n$ 
   $l_{i1} \leftarrow u_{i1}/l_{11}$ 
  for  $j = 2, \dots, i$ 
     $tmp \leftarrow 0$ 
    for  $k = 1, \dots, j - 1$ 
       $tmp \leftarrow tmp + l_{jk}l_{ik}$ 
    end
    if  $j = i$ 
       $l_{ii} \leftarrow \sqrt{(u_{ij} - tmp)}$ 
    else
       $l_{ij} \leftarrow (u_{ij} - tmp)/l_{jj}$ 
    end
  end
end

```

There are two interesting features of Algorithm 13.11 related to the inner most loop. If  $L$  is actually stored as matrix, the inner most loop is the inner product of row  $i$  with row  $j$ . The steps in the innermost loop over  $k$  are not easily done in parallel because each step must access  $tmp$  which is changing. Therefore, vector computers will not do well on this type of operation. See, for example, Dowd [130]. Also, important in terms of implementation, is the programming language used. FORTRAN stores matrices by column, whereas C stores matrices by rows. Thus, coding this algorithm in C results in more localized memory access. These implementation issues are discussed further in Section 13.7 and a different implementation of (13.34) is given later in Algorithm 13.17.

In the discussion so far we have calculated the Cholesky factorization  $L$ , row by row. George and Liu [178] refer to this as the *bordering method* or *row method*. There are other strategies. For example, write  $M$ , the matrix to be factored as

$$M = \begin{bmatrix} s & u^T \\ u & C \end{bmatrix}.$$

Then  $M = LL^T$  implies  $l_{11} = \sqrt{s}$  and  $l_{11}l_{i1} = u_i$  for  $i = 2, \dots, m$  which gives

$$l_{i1} = u_i/l_{11} = u_i/\sqrt{s}, \quad i = 2, \dots, n. \quad (13.36)$$

Then

$$M = \begin{bmatrix} \sqrt{s} & 0 \\ u/\sqrt{s} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \hat{C} \end{bmatrix} \begin{bmatrix} \sqrt{s} & u^T/\sqrt{s} \\ 0 & I_{n-1} \end{bmatrix} \quad (13.37)$$

where

$$\hat{C} = C - uu^T/s. \quad (13.38)$$

If  $\hat{L}\hat{L}^T = \hat{C}$  is the Cholesky factorization of  $\hat{C}$ , it follows from (13.37) that

$$M = \begin{bmatrix} \sqrt{s} & 0 \\ u/\sqrt{s} & \hat{L} \end{bmatrix} \begin{bmatrix} \sqrt{s} & u^T/\sqrt{s} \\ 0 & \hat{L}^T \end{bmatrix}$$

The second column of  $L$  is calculated by applying (13.36) and finding the first column of the modified  $(n-1) \times (n-1)$  matrix  $\hat{C}$ . This process is repeated in turn for each column of  $L$ . This method is called the *outer-product* or *submatrix* Cholesky factorization. Finally, there is a third method, where column  $j$  is calculated based on the previous calculation of columns  $1, \dots, j-1$ . This is called the *inner-product* or *column* Cholesky factorization. Although each method will produce the same unique Cholesky factorization of  $M$ , an efficient implementation of the different methods is highly dependent on computer architecture aspects such as cache access, virtual memory, vectorization, and pipelining. See Ng and Peyton [357] for a good discussion. Several of this issues are discussed in Section 13.7.

In the barrier algorithms, the Cholesky factorization of a matrix  $M = ADA^T$  is used in much the same way the *LU* factorization of a basis matrix  $A_B$  is used in the revised simplex algorithm. In Chapter 8 various barrier algorithms are proposed. The common denominator of these algorithms is the necessity of solving a system of the form  $(ADA^T)d = r$  for  $d$  which is a direction in primal or dual space. Rather than calculate an explicit inverse  $(ADA^T)^{-1}$ , the Cholesky factorization  $ADA^T = LL^T$  is used to solve the system  $LL^Td = r$  in a two step process. First, a solution  $\bar{y}$  to system  $Ly = r$  is found using forward substitution. Next, a direction  $\bar{d}$  is found to the system  $L^Td = \bar{y}$  using backward substitution.

## 13.5 SYMBOLIC CHOLESKY FACTORIZATION

As with *LU* decomposition, it is essential to take advantage of sparsity because fill-in is also a problem with Cholesky factorization. To motivate this, consider the outer-product Cholesky factorization using the formula  $\hat{C} = C - uu^T/s$ . Barring a “lucky” arithmetic cancelation, element  $(i,j)$  is nonzero in  $\hat{C}$  if  $(i,j)$

is nonzero in  $C$ , or  $u_i$  and  $u_j$  are nonzero. Define

$$\text{Nonz}(M) := \{(i, j) \mid a_{ij} \neq 0, i \neq j\}.$$

If  $M$  is factored into  $LL^T$  using one of the Cholesky factorization algorithms, the *filled matrix*,  $F(M)$  of  $M$  is defined by the matrix sum  $F(M) := L + L^T$ . The corresponding nonzero structure is

$$\text{Nonz}(F(M)) = \{(i, j) \mid l_{ij} \neq 0, i \neq j\}$$

Assuming no numerical cancelation during the factorization,  $\text{Nonz}(M) \subseteq \text{Nonz}(F(M))$  and the *fill of matrix  $M$*  is defined by

$$\text{Fill}(M) := \text{Nonz}(F(M)) \setminus \text{Nonz}(M). \quad (13.39)$$

Since the Cholesky factorization is unique,  $\text{Nonz}(F(M))$  is well defined. In order to minimize the fill-in we model the symmetric matrix as an undirected graph.

Given an  $n \times n$  symmetric matrix  $M$ , define the graph of  $M$  by  $G(M) = (V(M), E(M))$  where  $V(M) = \{x_1, \dots, x_n\}$  is the vertex set and  $E(M)$  is the edge set such that  $(x_i, x_j) \in E(M)$  if and only if  $a_{ij} = a_{ji} \neq 0$ . See Appendix C for an introduction to the basics of graph theory.

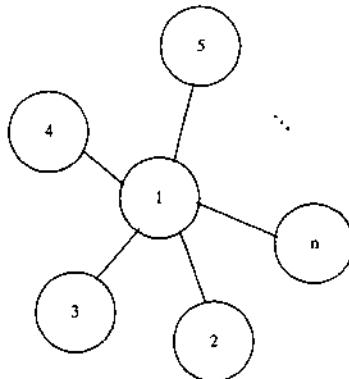
**Example 13.12** Consider the matrix

$$M = \begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & 0 \\ \vdots & & \ddots & 0 \\ * & 0 & \cdots & * \end{bmatrix}$$

The corresponding graph appears in Figure 13.1. If  $M$  is  $n \times n$  then  $|\text{Nonz}(M)| = 3n - 2$ ,  $|\text{Nonz}(F(M))| = n^2$  and  $|\text{Fill}(M)| = n^2 - 3n + 2$  which is the maximum possible for  $M$ . Now permute the first and last row  $M$  and the first and last column of  $M$ . In this case,  $|\text{Nonz}(F(M))| = 3n - 2$  and  $|\text{Fill}(M)| = 0$ .

Example 13.12 suggests that permuting the matrix to be factored can result in tremendous savings of nonzero fill-in. Indeed, one would like to find a permutation matrix  $P$  which is the solution to the problem

$$\min_P |\text{Fill}(PMP^T)|.$$

**Figure 13.1** Star Graph for Matrix M

In all barrier algorithms, Cholesky factorization is applied to a matrix with the structure  $ADA^T$ . At each iteration of a barrier algorithm, only the nonzero diagonal matrix  $D$  changes,  $A$  remains constant. This implies that the nonzero structure of  $ADA^T$  does not change from iteration to iteration which means that it is worthwhile to expend effort finding a good matrix ordering. Unfortunately, the decision problem “given an integer  $q$  and positive definite matrix,  $M$ , does there exist a permutation matrix  $P$  such that  $\text{Fill}(PMP^T) \leq q$ ” is  $\mathcal{NP}$ -complete (See Rose and Tarjan [386]) and so heuristics are used in practice. We describe the *minimum degree* heuristic of Tinney [423]. The intuition behind this method is illustrated in Example 13.12. The  $u$  vector for this example has all nonzero elements so  $uu^T$  is an  $n - 1 \times n - 1$  matrix which is completely dense. Then, from (13.38)  $\hat{C}$  is perfectly dense, and the associated graph  $G = (X(\hat{C}), E(\hat{C}))$  is a clique. Now permute the node numbering and exchange nodes 1 and  $n$ . This corresponds to a permutation matrix  $P$  defined by exchanging rows 1 and  $n$  in the identity matrix  $I_n$ . Then for the matrix  $PMP^T$ ,  $u$  is now a unit vector with the one in the last component and  $\hat{C}$  in the permuted matrix has the same nonzero pattern as  $C$ . The graph  $(X(\hat{C}), E(\hat{C}))$  is also a *star graph*. The graph  $(X(\hat{C}), E(\hat{C}))$  defined from the matrix

$$M = \begin{bmatrix} s & u^T \\ u & C \end{bmatrix}$$

by  $\hat{C} = C - uu^T/s$  is the *elimination graph* of  $(V(M), E(M))$ .

**Proposition 13.13** *In the elimination graph  $(X(\hat{C}), E(\hat{C}))$  of the matrix*

$$\hat{C} = C - uu^T/s$$

there is a clique with vertex set  $\{i \mid u_i \neq 0\}$ .

Proposition 13.13 is stating that in the graph of matrix  $\hat{C}$ , there is an edge between vertices  $i$  and  $j$ , if in the graph of  $C$ , vertices  $i$  and  $j$  are adjacent to vertex corresponding to the first column of  $C$ . That is, there is a nonzero element in rows  $i$  and  $j$  of the first column of  $C$ . This suggests at step  $k$  of Cholesky factorization a “greedy” or locally optimal strategy of selecting the node  $x_i$  with the minimum degree (number of adjacent nodes), i.e. at each step, order the matrix so the first column of  $C$  has the minimum number of nonzero elements. This results in the smallest clique being formed in the elimination graph. Note the close analogy with Markowitz pivot selection strategy developed earlier in Subsection 13.2.2.

#### Algorithm 13.14 (Minimum Degree Ordering Algorithm)

```

 $G_0 \leftarrow (V(M), E(M))$ 
for  $i = 1, \dots, n$ 
    select node  $x_k$  in  $G_{i-1}$  of minimum order
    form new elimination graph  $G_i = (V(\hat{C}), E(\hat{C}))$ 
     $p_{ik} \leftarrow 1$  in row  $i$  of permutation matrix  $P$ 
end

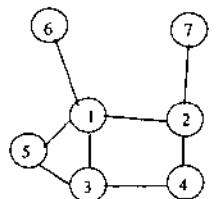
```

**Example 13.15** This example is taken from George and Liu [178].

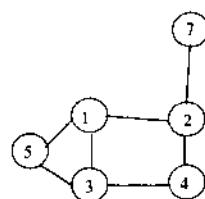
$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & & * \\ * & * & * & * & \\ * & * & & & \\ * & * & * & & \\ * & & & * & \\ * & & & & * \end{bmatrix}$$

The sequence of elimination graphs are in Figure 13.2.

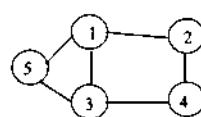
There are other ordering algorithms in the literature. See, for example, George and Liu [179]. Liu [291] modified the basic minimum degree ordering algorithm with the multiple minimum degree algorithm and this was the default in the OB1 algorithm of Lustig, Marsten, and Shanno [302]. There are four possible orderings in the CPLEX barrier implementation.

**Figure 13.2** Minimum Degree Order Algorithm

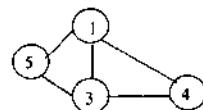
$i = 1, k = 6$



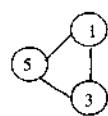
$i = 2, k = 7$



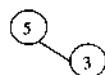
$i = 3, k = 2$



$i = 4, k = 4$



$i = 5, k = 1$



$i = 6, k = 3$



$i = 7, k = 5$

## 13.6 STORING SPARSE MATRICES

It is impractical to store the constraint matrix  $A$ , of a large linear program as a two dimensional array. This is because the  $A$  matrix is always very sparse for real world problems and performing arithmetic operations on zero elements is a waste. Also, storing even a moderate sized problem, say 1000 rows and 10000 variables in dense form as a two dimensional array would require approximately 40 megabytes of RAM (assume each element of the array requires four bytes). Instead, the nonzeros of the  $A$  matrix are stored in a single dimensional array. For example, storing the nonzero elements of the constraint  $A$  matrix by column, would require three arrays: a  $a()$ ,  $point()$ , and  $inrow()$ . The array  $point()$  is a pointer array, with element  $point(i)$  pointing to the start of the nonzero elements of column  $i$  in the array  $a()$  and  $inrow()$ . The nonzero elements for column  $i$  are stored in  $a(point(i))$  through  $a(point(i+1) - 1)$  and the row numbers are stored in  $inrow(point(i))$  through  $npoint(point(i+1)-1)$ .

**Example 13.16** Consider the following matrix.

$$A = \begin{pmatrix} 0 & 1.5 & 0 & 7.7 & 0 \\ 2.0 & 6.7 & 1.1 & 7.7 & 0 \\ 7.5 & 0 & 1.1 & 10 & 1 \\ 0 & 0 & 1.1 & 12 & 0 \\ 3.1 & 0 & 0 & 1.9 & 0 \end{pmatrix}$$

Then

$$\begin{aligned} a() &= 2.0, 7.5, 3.1, 1.5, 6.7, 1.1, 1.1, 1.1, 7.7, 7.7, 10, 12, 1.9, 1 \\ point() &= 1, 4, 6, 9, 14 \\ inrow() &= 2, 3, 5, 1, 2, 2, 3, 4, 1, 2, 3, 4, 5, 3 \end{aligned}$$

It is assumed in this storage scheme that the columns are stored in order. That is, the nonzero elements for column  $i + 1$  are stored in  $a()$  immediately after the nonzero elements for column  $i$ . If this is not the case, then an additional array  $length(i)$  is required which records the number of nonzero elements in column  $i$ . The nonzero elements of column  $i$  are stored between  $a(point(i))$  and  $a(point(i) + length(i) - 1)$ . The logic is similar for storing  $A$  by row.

Not only are most real world problems very sparse, but the number of *distinct* nonzero elements is relatively small compared with the number of nonzero elements. For example, in many problems many nonzero elements are  $\pm 1$ . Kalan [256] proposed a method where the distinct nonzero elements are stored in a

table and  $a(j)$  points to the table address of the nonzero element. This allows for  $a(j)$  to be stored as an integer (perhaps using only one or two bytes) rather than a real number. Unlike today, in the early days of linear programming, high speed random access memory was a scarce commodity and linear programming codes often used various techniques for storing more than one number in a word. See, for example, Day [118]. With memory so cheap and plentiful, some linear programming codes store the constraint matrix both by column and row. Reasons for doing so are discussed in the next section.

## 13.7 PROGRAMMING ISSUES

In this chapter we are concerned with taking advantage of the sparsity of a linear program. It is impossible to take advantage of sparsity without taking care to carefully code the algorithms. In this section we consider some coding issues which are relevant to simplex and barrier implementations of sparse systems.

One of the most common calculations in large scale linear programming is the solution of the system  $Lw = c$  or  $Uz = c$  where  $L$  ( $U$ ) is lower (upper) triangular. These calculations appear in Gaussian elimination, Cholesky factorization using the row method, pricing and column updates.

### 13.7.1 SAXBY and Dot Product

Refer back to Algorithm 13.11, the row Cholesky factorization method. This requires the solution of the system  $Lw = u$ . The inner most loop is an inner product calculation. Consider the same algorithm, but a different loop organization.

#### Algorithm 13.17 (Row- $ikj$ Cholesky Factorization,)

```

 $l_{11} \leftarrow \sqrt{u_{11}}$ 
for  $i = 2, \dots, n$ 
    for  $k = 1, \dots, i - 1$ 
         $l_{ik} \leftarrow u_{ik}/l_{kk}$ 
        for  $j = k + 1, \dots, i$ 
             $u_{ij} \leftarrow u_{ij} - l_{jk}l_{ik}$ 
        end
    end
end

```

$$l_{ii} \leftarrow \sqrt{u_{ii}}$$

end

In the inner loop of Algorithm 13.17,  $l_{ik}$  does not change with the loop index,  $j$ , so this operation consists of multiplying a vector by a scalar and adding it to another vector. This operation is known as either a *SAXPY* (single precision  $A$  times  $X$  plus  $Y$ ) or *DAXPY* (double precision  $A$  times  $X$  plus  $Y$ ) depending on the precision. This operation can be done very fast on vector processing machines. For each value of  $j$  there are three memory operations (two loads and a store) and two floating point operations (add and multiply) and on Cray-like machines this can be done in one clock cycle. See George, Heath, and Liu [177] for a discussion of which loop organizations are more amenable to parallelization.

### 13.7.2 Gather and Scatter

When solving  $Lw = c$  or  $Uz = c$  the matrices  $L$  and  $U$  are not stored as two dimensional arrays but are stored by column or row as described in Section 13.6. This has significant implications for how one takes the inner product of two sparse vectors or performs a SAXPY. For example, in the inner loop of Algorithm 13.17 consists of the SAXPY  $u_{ij} \leftarrow u_{ij} - l_{jk}l_{ik}$ . However, when working with sparse systems,  $m()$  and  $l()$  are stored as one dimensional arrays and a SAXPY is often best done by *scattering* a sparse array to a dense work array.

To illustrate the scatter process, assume the matrix is stored in the array  $a()$  in row form. The nonzero elements of row  $i$  are stored between  $a(point(i))$  and  $a(point(i) + length(i) - 1)$  and the column index is contained in the array  $incol()$  between  $point(i)$  and  $point(i) + length(i) - 1$ .

Consider the generic operation of replacing row  $i$  with row  $i$  plus  $\alpha$  times row  $j$ . First, the scatter operation to a dense vector.

#### Algorithm 13.18 (Sparse SAXPY - Scatter)

```
/* scatter into dense array x() */
/* initialize dense array to 0 */
for k = 1, ..., n
```

```

     $x(k) \leftarrow 0$ 
end
/* put nonzero elements of row  $i$  into dense array  $x()$  */
for  $k = point(i), \dots, point(i) + length(i) - 1$ 
     $x(incol(k)) \leftarrow a(k)$ 
end
/* replace row  $i$  with row  $i + alpha$  times row  $j$  */
for  $k = point(j), \dots, point(j) + length(j) - 1$ 
     $x(incol(k)) \leftarrow x(incol(k)) + alpha*a(k)$ 
end

```

Once row  $i$  is recalculated it is then restored, or *gathered* into sparse form. One way to do this is to restore it starting with position  $a(point(i))$ . However, there is a problem with doing this. The number of nonzero elements in the updated row  $i$  could increase which would make it necessary to adjust  $a()$  and  $incol()$  for all rows after  $i$ . An easier solution is to make the array  $a()$  large and put the updated row  $i$  after the last nonzero which is stored in position  $a(last)$ . It is also necessary to zero out the dense work array. The gather process is illustrated in Algorithm 13.19

#### Algorithm 13.19 (Sparse SAXPY - Gather)

```

/* gather back into sparse array  $a()$  */
/* assume the last nonzero is stored in  $a(last)$  */
/* replace row  $i$  with row  $i + alpha$  times row  $j$  */
/* also zero out the dense vector */
kount  $\leftarrow 1$ 
point( $i$ )  $\leftarrow last + 1$ 
for  $k = point(j), \dots, point(j) + length(j) - 1$ 
     $a(last + kount) \leftarrow x(incol(k))$ 
     $incol(last + kount) \leftarrow incol(k)$ 
     $x(incol(k)) \leftarrow 0$ 
    kount  $\leftarrow kount + 1$ 
end
for  $k = point(i), \dots, point(i) + length(i) - 1$ 
    if  $x(incol(k)) \neq 0$  {
         $a(last + kount) \leftarrow x(incol(k))$ 
         $incol(last + kount) \leftarrow incol(k)$ 
         $x(incol(k)) \leftarrow 0$ 
        kount  $\leftarrow kount + 1$  }

```

```

end
length(i) ← kount - 1
last ← last + length(i)

```

One problem with working with sparse arrays is that the compiler/hardware configuration may not be able to vectorize the SAXPY operation if it operates with indexed vectors and not dense vectors. This is because the compiler may not be able to detect if a *recurrence* exists. A recurrence exists when a value calculated in one iteration of a loop is referenced in another iteration. There are several alternatives for handling indexed SAXPYs. First, if the user knows that recurrence is not a problem, then a compiler directive can be used. For example, in Convex FORTRAN the compiler directive **FORCE\_VECTOR** will force the compiler to vectorize the loop. Of course this is compiler/hardware dependent which is a problem. A second solution is to try to use dense SAXPYs. When implementing Cholesky factorization algorithms, authors such as Jung, Marsten, and Saltzman [255] and Ng and Peyton [357] use the concept of *supernodes* in order to reduce the number of sparse SAXPYs and replace with dense SAXPYs. Finally, see Chapter 2 of Duff, Erisman, and Reid [132] for a good discussion of programming vector operations use sparse data structures.

There are several places in the revised simplex algorithm where it is important to take advantage of sparse storage and sparse SAXPY operations. In Step 3 of revised simplex, the pivot column is updated by solving the system  $A_B \bar{a}^q = LU \bar{a}^q = a^q$ . After the FTRAN,  $\hat{a} = L^{-1}a^q$ , a back solve is used to solve the system  $U \bar{a}^q = \hat{a}^q$ . In order to solve this system it is advantageous to store the sparse matrix  $U$  by column thus making it easy to code the solution of  $U \bar{a}^q = \hat{a}^q$  so that the inner loop is a SAXPY. See Algorithm 13.4.

In Step 5 of revised simplex, it is necessary to find row  $i^*$  of  $A_B^{-1}$ . This is done by solving the system

$$(z^{i^*})^T A_B = (e^{i^*})^T LU = (e^{i^*})^T.$$

This requires using a forward solve to find  $y^T U = (e^{i^*})$ . In this case, it is advantageous to store the sparse matrix  $U$  by row, thus making it easy to code the solution of  $y^T U = (e^{i^*})^T$  so that the inner loop is a SAXPY. Due to inexpensive fast memory many codes often store  $A$  by both row and column.

In Step 6 of revised simplex, the reduced costs are updated using the formula

$$\bar{w}_N^T \leftarrow \bar{w}_N^T - \alpha_D (z^{i^*})^T A_N.$$

If  $A_N$  is stored in row form, then it is most efficient to make this calculation with a nested loop. The outer loop is over the rows and the inner loop over the nonbasic columns. If a component of  $z^*$  is zero, then the inner loop is skipped for that component. The inner loop is a SAXPY where a row is multiplied by the scalar  $\alpha_D z_j^*$  and then added to the working vector of reduced costs. Bixby [55] reports that the speedup for calculating  $\bar{w}_N$  in this manner is substantial. In general, the number of nonbasic variables far exceeds the number of rows of  $A_N$  so in this loop organization the inner loop is much longer than the outer loop. Good loop organization is the topic of the next subsection.

### 13.7.3 Loop Optimization and Locality of Reference

Most of the work in any simplex or barrier algorithm occurs within a loop or nested loops. Therefore, it is important to code the loops for efficient implementation. When nested loops are present, if possible, make the inner loop long. Consider the following example.

**Example 13.20 (Loop Interchange)** Consider the nested loops

```
for i = 1, ..., 1000
    for j = 1, ..., 10
        calculation
    end
end
```

and the alternative ordering

```
for j = 1, ..., 10
    for i = 1, ..., 1000
        calculation
    end
end
```

In the first case, there are  $1 + 1000 = 1001$  loop setups, in the second case there are  $1 + 10 = 11$  loop setups.

Two other reasons for making the innermost loop long are that: 1) parallel or vector processor computers can better exploit long loops, and 2) a long loop is a candidate for *loop unrolling*. The basic idea is to reduce the number of iterations and loop overhead. Consider the following example from Dowd [131].

**Example 13.21 (Loop Unrolling)** Consider the simple loop

```
s ← 0
for i = 1, ..., n
    s ← s + a(i)*b( i)
end
```

This loop can be rewritten as

```
s0 ← 0
s1 ← 0
for i = 1, ..., n, 2
    s0 ← s0 + a( i) * b( i)
    s1 ← s1 + a( i + 1)* b( i + 1)
end
s ← s0 + s1
```

Finally, avoid any unnecessary clutter inside the loops. For example, an expression which does not change with the iteration count should be moved outside the loop. Similarly keep **if** statements outside the loop if possible. We highly recommend the book by Dowd [130] for those wishing to learn more about compiler and hardware issues affecting numerical computation.

Related to loop organization is the *locality of reference* problem. Located between the system RAM (random access memory) and processor (CPU) is the *processor cache*. (Modern systems generally have two caches, an L1 cache physically on the processor chip and an L2 cache external to the processor chip.) This is designed to allow the processor to execute instructions on the needed data as quickly as possible without resorting to the relatively slow access of RAM. The connection between the cache and processor is faster than the connection between the cache and RAM. When a processor needs information, the processor tries to fetch it from the cache. If that information is not in the cache, a cache miss occurs, and a batch of data is moved from RAM into the cache.

The *line width* is the number of bytes of data moved into the cache from RAM. If the application requests data in nearby memory location, then the number of cache misses is going to be minimized and the data the processor needs is accessed quickly.

Consider, for example, updating the dual variables. The formula is

$$\bar{w}^T \leftarrow \bar{w}_N^T - \alpha_D(z^{i^*})^T A_N.$$

If  $A_N$  is stored by column then it is not necessarily the case that nonbasic variable  $i$  and  $i+1$  correspond to columns of  $A$  that are adjacent. This means updating the reduced costs by column could result in a number of cache misses. However, assume  $A_N$  is stored by row. Calculating  $\alpha_D(z^{i^*})^T A_N$  by row results in long vectors being brought into memory and the elements of these long vectors accessed sequentially resulting in few cache misses. See Wilson [457] for the effect of the cache on matrix computations. This illustrates how important it is to consider the underlying hardware when coding algorithms.

### 13.7.4 Language Issues

The two most popular languages for scientific computing are FORTRAN and C. With powerful optimizing compilers, a natural question is: "does it make any difference which language is used?" There are arguments either way. In terms of programming, there are some clear advantages for C over FORTRAN 77. The C programming language allows for pointer variables and structures. Older versions of FORTRAN compilers did not allow dynamic memory allocation. Lustig [298] has demonstrated speed improvements by using C pointer variables that reference actual locations in memory, as opposed to an integer pointer array such as *point*( $i$ ) which points to an array location. Finally, it is important to note that when working with two dimensional arrays, in order to reduce memory access time, it is best to access the array by column in FORTRAN and by row in C.

## 13.8 COMPUTATIONAL RESULTS: BARRIER VERSUS SIMPLEX

Which is superior – barrier or simplex – for the solution of large, sparse problems? This is not easy to answer. At least initially, there may have been some bias in the barrier results because researchers on barrier algorithms were given

test problems for which simplex was having trouble. Perhaps the most succinct statement is given by Lustig, Marsten, and Shanno [302]

*... for small problems, simplex codes are far more efficient than interior point codes. An immediate question arises as to what is small. In our experience, any problem where the sum of the number of rows plus the number of columns is less than 2000 is considered small and much more likely to be solved faster by simplex codes. For moderate problems, when the number of rows plus the number of columns is less than 10,000, results get more interesting. Here, good simplex codes and interior point codes compete more evenly in that, overall, no one method shows a clear advantage. On specific models, however, the differences can be very dramatic depending on the structure of the nonzeros in the matrix A.*

Certainly, the development of barrier algorithms has spurred researchers to go back and improve simplex based codes using the latest sparse matrix technology. In the January, 1995 meeting of the Computer Science Technical Subsection of INFORMS, Bixby reported for a test set of 35 problems, that with version 4.0 of CPLEX running on an SGI R8000, simplex is faster on 8-10 problems and barrier faster on 25-27 problems.

Fortunately for researchers, there are several test sets of realistic, well documented linear programs which can be used to evaluate linear programming methodology. See the **NETLIB** test set at the following URL.

<http://www.mcs.anl.gov/home/otc/Guide/TestProblems/LPtest/index.html>

## 13.9 CONCLUSION

Regardless of which linear programming algorithm is used, revised simplex or barrier, careful implementation is crucial. A key development in the improving the performance of revised simplex is the sparse *LU* update at each pivot which extends the earlier work of Markowitz [309]. For good summaries of sparse *LU* updates see Murtagh [348] and Suhl and Suhl [415]. Similarly, a sparse Cholesky factorization is crucial to the success of any barrier algorithm. Several good summary papers include Jung, Marsten, and Saltzman [255] and Ng and Peyton [357]. For an understanding of the computer science aspects of numerical linear algebra see Dongarra, Gustavson, and Karp [129] and Dowd [130].

## 13.10 EXERCISES

- 13.1 Prove that if  $A$  is an  $m \times n$  matrix with rank  $m$ ,  $D$  is a diagonal matrix with positive diagonal elements, then  $M = ADA^T$  is a positive definite matrix. A square matrix  $M$  is positive definite if  $x^T M x > 0$  for all nonzero  $x \in \mathbb{R}^n$ .
- 13.2 Prove that if  $M$  is an  $n \times n$  positive definite matrix and  $B$  is an  $n \times n$  matrix of rank  $n$ , then  $B^T M B$  is a positive definite matrix.
- 13.3 Prove Proposition 13.10.
- 13.4 Prove Proposition 13.13.
- 13.5 In Example 13.21 we assume that  $n$  is divisible by two. How can one *precondition* the loop to make this assumption valid?
- 13.6 Prove Lemma 13.2.
- 13.7 Prove Proposition 13.3.
- 13.8 Explain why it is better to solve system (13.12) through back substitution, rather than finding  $U^{-1}$  and calculating  $x = U^{-1}\bar{y}$ .
- 13.9 Assume that in the  $LU$  decomposition of the basis matrix, that  $U$  is stored by column. When  $U$  is updated to  $\hat{U}$  in (13.23) during a simplex pivot explain how the data structures would be modified to replace  $U$  with  $\hat{U}$ . Now answer the question assuming  $U$  is stored by row.
- 13.10 Finish working to optimality Example 13.8.
- 13.11 Assume the sparse upper triangular matrix  $U$  is stored by column in a sparse format. Write a C or FORTRAN subroutine for solving the system  $Ua^q = \hat{a}^q$ .
- 13.12 Row  $i^*$  of the inverse of the basis matrix  $A_B$  is required in order to update the reduced costs. If  $A_B = LU$ , row  $i^*$  of  $A_B^{-1}$  is found by first solving the system  $y^T U = (e^{i^*})^T$ . Describe how to take advantage of the fact that  $e^{i^*}$  is a unit vector in programming the forward substitution algorithm.
- 13.13 Find the  $LU$  decomposition of the basis matrix given in Example 5.11. In general, for a matrix of this structure with  $2n + 1$  rows and columns, how many nonzero elements are required in the  $LU$  decomposition.
- 13.14 Apply the bump reduction of idea to (13.29) in Example 13.8 and permute the matrix into upper triangular form.

## NETWORK FLOW LINEAR PROGRAMS

### 14.1 INTRODUCTION

In this chapter we study a class of linear programs called *network flow problems*. These problems are important for several reasons. Many important applications give rise to linear programming models where all, or a large portion, of the constraints have a network flow structure. The constraint matrix of a network flow linear program has a structure that enables these linear programs to be optimized very quickly. In addition, the associated polyhedron is integer which means that no enumeration is required in order to find integer solutions. Thus, not only can the linear programming relaxation of a network problem be solved quickly, no enumeration is necessary.

In the Section 14.2 we study the algebraic properties of the constraint matrix of network flow problems. In Section 14.3 we show how to specialize the simplex algorithm to linear programs with this structure. In Section 14.4 we study a number of important applications which can be modeled as minimum cost network flow linear programs. In Section 14.5 we examine classes of problems that are “almost,” but not quite, network linear flow problems. An important property of minimum cost network flows is that the associated polyhedron is integer. In Section 14.6, we study a broader class of problems, called *totally dual integer problems* that include network flow problems and have integer polyhedra. Concluding remarks are given in Section 14.7. Exercises are provided in Section 14.8.

The *minimum cost network flow linear program*, (*MCNF*), is a linear program defined over a directed graph  $G = (V, \mathcal{A})$  where  $V$  is the vertex set and  $\mathcal{A}$  is the set of arcs which are ordered pairs of elements of  $V$ . We assume: 1) there are no loop arcs  $(i, i)$ , 2) there are no duplicate arcs, and 3) that  $G$  is a connected

graph. Associated with each vertex  $l \in V$  is a net demand  $b_l$ . This demand may be positive, in which case it is called a *source* or *supply* vertex, it may be zero in which case it is called a *transshipment* vertex, or it may be negative in which case it is called a *sink* or *demand* vertex. Associated with each arc  $(i, j)$  is  $c_{ij}$ , the marginal cost of flow on the arc. If  $x_{ij}$  denotes the flow on arc  $(i, j)$  the minimum cost network flow problem is

$$\min \quad \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \quad (14.1)$$

$$(MCNF) \quad \text{s.t.} \quad \sum_{(i,l) \in \mathcal{A}} x_{il} - \sum_{(l,j) \in \mathcal{A}} x_{lj} = b_l, \quad l \in V \quad (14.2)$$

$$x_{ij} \geq 0, \quad (i, j) \in \mathcal{A} \quad (14.3)$$

The constraint matrix (14.2) of  $(MCNF)$  has a column for each  $(i, j) \in \mathcal{A}$  and in each column there is a  $-1$  in the row corresponding to vertex  $i$  and a  $+1$  in the row corresponding to vertex  $j$ . This is called a *node-arc* or *vertex-arc incidence matrix* for the graph  $G$ . This problem is also called the transshipment problem.

In formulation  $(MCNF)$ , we are assuming that there is infinite flow capacity on every arc. In several of the application problems considered later in the chapter, there are simple upper bounds  $x_{ij} \leq h_{ij}$  constraining the flow on arc  $(i, j)$  to be less than the arc capacity of  $h_{ij}$ . Fortunately, these upper bound constraints do not destroy the important properties of the constraint matrix. These properties are the topic of the next section.

## 14.2 TOTALLY UNIMODULAR LINEAR PROGRAMS

One of the important properties of the minimum cost network flow linear program is that the associated polyhedron is integer. The integrality property is due to the special structure of its constraint matrix. The constraint matrix is an example of a totally unimodular matrix. A nonsingular matrix  $A$  is *unimodular* if it is integer and has a determinant of  $\pm 1$ . A matrix  $A$  is *totally unimodular* if every square submatrix of  $A$  has determinant  $\pm 1$  or 0.

**Lemma 14.1** *If  $A$  is an  $m \times n$  totally unimodular matrix then:*

1.  $A^T$  is totally unimodular;

2.  $(A, I_m)$  is totally unimodular;
3. pivot operations preserve total unimodularity of a matrix, that is, if element  $a_{ik} \neq 0$  is used as a pivot element and  $\bar{A}$  is the matrix that results and  $B$  is a square submatrix of  $A$ , then for the corresponding square submatrix  $\bar{B}$  in  $\bar{A}$ , then either  $|\det(\bar{B})| = 0$ , or  $|\det(B)| = |\det(\bar{B})|$ ;
4.  $(A, A)$  is totally unimodular.

**Lemma 14.2** Assume  $A$  is a totally unimodular matrix. If variable  $x_i$  is projected out of the system  $Ax \geq b$ , then the resulting system in  $\mathbb{R}^{n-1}$ ,  $\tilde{A}\tilde{x} \geq \tilde{b}$ ,  $\tilde{A}$  is also totally unimodular.

**Proof:** Without loss, assume variable  $x_1$  is projected out. First show that  $\tilde{A}$  has only nonzero elements of  $\pm 1$ . The projected system is created by adding together every pair of rows with a +1 and -1 in column one. If adding two such rows together results in  $\pm 2$  then  $A$  must contain (possibly after row and column permutations) one of the two submatrices:

$$\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix}. \quad (14.4)$$

However, both submatrices in (14.4) have a determinant of  $\pm 2$ . Thus,  $\tilde{A}$  has entries of  $\pm 1, 0$ .

Now show that every nonsingular submatrix of  $\tilde{A}$  with rank two or greater has a determinant of  $\pm 1$ . Assume not, and that there is a  $k \times k$  nonsingular matrix  $\tilde{B}$  contained in  $\tilde{A}$  with a determinant that is not  $\pm 1$ . Without loss, assume  $\det(\tilde{B}) = \pm 2$ . (See Statement 1 in Camion [79].) Let  $I$  index the rows of  $A$  that were used to generate the rows in  $\tilde{B}$  using projection. The rows of  $\tilde{B}$  are generated by adding rows of the matrix  $\bar{A}$  where  $\bar{A}$  is the submatrix of  $A$  obtained by deleting the columns of  $A$  which do not correspond to columns in  $\tilde{B}$  and deleting rows not indexed by  $I$ . Thus, the rows of  $\tilde{B}$  are in the span of the rows of  $\bar{A}$  over the vector space with field  $\{0, 1\}$  and addition modulo 2. By hypothesis,  $\det(\tilde{B}) = \pm 2$  so the row vectors of  $\tilde{B}$  are linearly dependent over this vector space. This implies there is a set  $S_1$  of strictly less than  $k$  rows of  $\bar{A}$  which have the same span as the rows of  $\tilde{B}$  over the vector space with field  $\{0, 1\}$  and addition modulo 2.

However,  $\tilde{B}$  is nonsingular, which implies it has rank  $k$  in the vector space defined over the field of rational numbers. Thus, there is a set  $S_2$  of  $k$  linearly

independent rows (over the rational field) from  $\bar{A}$  which span the rows of  $\tilde{B}$  over the vector space defined over the field of rational numbers and  $S_1 \subset S_2$ . Let  $\bar{B}$  denote the submatrix of  $\bar{A}$  consisting of the rows indexed by  $S_2$ . Since  $S_1 \subset S_2$ ,  $\bar{B}$  has rank less than  $k$  over the vector space with field  $\{0, 1\}$  and addition modulo 2. By hypothesis,  $A$  is totally unimodular so  $\bar{B}$  and all nonsingular submatrices of  $\bar{B}$  have determinant  $\pm 1$ . Then linear dependence over the field of rationals coincides with linear dependence over the field  $\{0, 1\}$ . This contradicts the fact that  $\det(\bar{B})$  is nonsingular since linear dependence over the two vector spaces coincides. Therefore, there cannot be a matrix  $\tilde{B}$  with determinant  $\pm 2$ .  $\square$

One of the most significant aspects of totally unimodular constraint matrices is that the underlying polyhedron is integer. A rational polyhedron is an *integer polyhedron* if every minimal face contains an integer vector. If the constraint matrix is full rank this implies that every extreme point is integer. Thus, the linear programming relaxation is integer and no enumeration is required.

**Theorem 14.3** *If  $A$  is a totally unimodular matrix and  $b, l$ , and  $h$  are integer vectors then*

$$P = \{x \in \mathbb{R}^n \mid Ax \geq b, x \geq l, x \leq h\}$$

*is an integer polyhedron.*

**Proof:** By repeated application of Lemma 14.1

$$\hat{A} := \begin{bmatrix} A \\ I_n \\ -I_n \end{bmatrix}$$

is totally unimodular. Matrix  $\hat{A}$  has rank  $n$  so it suffices to show each extreme point of  $P$  is integer. If  $\bar{x}$  is an extreme point of  $P$ , then  $\bar{x}$  is the solution to the system  $\bar{A}\bar{x} = \bar{b}$  where  $\bar{A}$  is a matrix obtained by deleting rows from  $\hat{A}$  and  $\bar{b}$  is a vector obtained by deleting components from the integer vector  $[b, l, -d]^T$ . The result now follows from part 3 of Lemma 14.1.  $\square$

Having the total unimodularity property is extremely important and therefore we want to be able to recognize totally unimodular matrices. In Theorem 14.4 we give five equivalent characterizations of totally unimodular matrices.

**Theorem 14.4** *If  $A$  is a matrix with  $\pm 1$ , and 0 entries, then the following conditions are equivalent:*

1.  $A$  is a totally unimodular matrix;
2. the polyhedron  $\{x \in \mathbb{R}^n \mid Ax \geq b, x \geq l, x \leq h\}$  has integer vertices for all integer  $b, l$ , and  $h$ ;
3. for any index set  $J$ , of columns of  $A$  there exists  $J_1, J_2$  such that  $J_1 \cup J_2 = J$ ,  $J_1 \cap J_2 = \emptyset$  and

$$\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| \leq 1, \quad i = 1, \dots, m,$$

that is, any collection of columns can be partitioned into two sets such that the row sum for set one minus the row sum for set two is a vector with  $\pm 1, 0$  components;

4. every square Eulerian submatrix of  $A$  is singular;
5.  $A$  does not contain the square matrix with determinant  $\pm 2$ .

**Proof:** In Theorem 14.3 we proved condition 1 implies condition 2. Show condition 2 implies condition 3. Let  $d$  be the characteristic vector of an arbitrary collection  $|J|$  of columns for  $A$ . Define the polyhedron

$$P := \{x \in \mathbb{R}^n \mid [\frac{1}{2}Ad] \leq Ax \leq [\frac{1}{2}Ad], 0 \leq x \leq d\}.$$

Clearly  $P \neq \emptyset$  since  $x = \frac{1}{2}d \in P$ . By hypothesis,  $P$  is an integer polyhedron. Since  $d$  is a 0/1 vector every vertex of  $P$  is a 0/1 vector. Let  $\bar{x}$  be an arbitrary vertex in  $P$ . For all  $j \in J$ , define  $y_j := d_j - 2\bar{x}_j = \pm 1, 0$ . Define  $J_1 := \{j \mid y_j = 1\}$  and  $J_2 := \{j \mid y_j = -1\}$ . Then

$$\sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} = \sum_{j \in J} a_{ij}y_j.$$

If component  $i$  of  $Ad$  is an even number then component  $i$  of  $A\bar{x}$  is an even number and for this component  $\sum_{j \in J} a_{ij}y_j$  is zero. If component  $i$  of  $Ad$  is an odd number, then component  $i$  of  $A\bar{x}$  is either component  $i$  of  $[\frac{1}{2}Ad]$  or component  $i$  of  $[\frac{1}{2}Ad]$ . In the former case,  $\sum_{j \in J} a_{ij}y_j = -1$  and in the later case  $\sum_{j \in J} a_{ij}y_j = +1$ . Therefore,  $J_1, J_2$  is the desired partition.

Show condition 3 implies condition 4. A square matrix  $B$  with elements  $0, \pm 1$  is *Eulerian* if all row and column sums are even. Let  $B$  be a nonsingular submatrix of  $A$ . Condition 3 implies there exists a nonzero vector  $x$  such that  $Bx$  is a vector with entries  $\pm 1, 0$ . If all of the rows of  $B$  contained an even

number of entries then  $Bx$  must actually be the zero vector. This cannot happen since  $x$  is nonzero and  $B$  nonsingular. Therefore  $B$  contains at least one row with an odd number of nonzero elements. Similarly for column sums.

Show condition 4 implies condition 5. Prove the contrapositive, not 5 implies not 4. Let  $B$  be a nonsingular submatrix of  $A$  with determinant  $\pm 2$ . Assume without loss, that there is no other submatrix with this property that has a rank less than the rank of  $B$ . Then every proper nonsingular submatrix of  $B$  has a determinant of  $\pm 1$ . Because the determinant of  $B$  is  $\pm 2$ , the rows of  $B$  over the vector space where the group addition is done modulo 2 and the field is  $\{0, 1\}$  are linearly dependent. However, since all proper nonsingular submatrices of  $B$  have determinant value  $\pm 1$ , linear dependence for any subset of columns of  $B$  in this vector space corresponds to linear dependence in the vector space over the field of real numbers. Thus, any proper subset of columns of  $B$  is linearly independent in the vector with group addition modulo 2 and the field  $\{0, 1\}$ . But all of the columns of  $B$  are not linearly independent over this vector space, so the sum of all columns of  $B$  must be zero modulo 2. This implies that the number of nonzero elements in each row is even.

Show condition 5 implies condition 1. Let  $B$  be a nonsingular submatrix of  $A$ . Pivot on the diagonal elements of  $B$  (after possibly permuting the rows so that the pivot element is nonzero). Since the nonzero elements of  $B$  are  $\pm 1$ , the first nonzero element other than  $\pm 1$  created must be a  $\pm 2$ . But this would imply there is a square submatrix of  $B$  with determinant  $\pm 2$ . Therefore, no nonzero elements other than  $\pm 1$  result during the pivoting process. This implies  $\det(B) = \pm 1$ .  $\square$

Condition 2 of Theorem 14.4 is due to Hoffman and Kruskall [232]. Condition 3 is due to Ghoulia-Houri [180]. See also Padberg [365]. Condition 4 is due to Camion [78] (and published earlier in a Ph.D. thesis) and 5 is attributed to R.E. Gomory by Camion [79]. Condition 3 is particularly useful in identifying totally unimodular matrices. The proofs given here are adopted from Schrijver [403].

**Example 14.5 (Minimum Cost Network Flow Problem)** *The constraint matrix for the minimum cost network flow problem given in (14.2) has exactly two nonzero elements per column. Take the transpose of this matrix and observe that the nonzero row sums are 0. Condition 3 of Theorem 14.4 is satisfied trivially by partitioning any collection  $J$  of columns into  $J_1 = J$  and  $J_2 = \emptyset$ . If integer upper (lower) bounds on arc flows are present then the constraint matrix for the upper (lower) bounds is an identity matrix so the constraint matrix for minimum cost network flow problem with capacity constraints on arc flows is*

*totally unimodular.* Given integer  $b_k$ ,  $k \in V$  the corresponding polyhedron is integer.

**Example 14.6 (Interval Matrices)** In Subsection 1.3.4 in Chapter 1 we gave a formulation for the multiproduct dynamic lot sizing problem. This formulation involved integer variables, capacities, and multiple products. A much simpler version of this problem involves a single product, no capacities and no fixed costs. In this simple version of the problem, the parameters are  $d_t$ , the demand in period  $t$ ;  $c_t$ , the marginal production cost in period  $t$ ; and  $h_t$ , the marginal cost of holding one unit of inventory at the end of period  $t$ . The only variable is  $x_t$ , the quantity produced in period  $t$ .

$$\begin{aligned} \min \quad & \sum_{t=1}^T c_t x_t + \sum_{t=1}^T h_t \sum_{l=1}^T (x_l - d_l) \\ \text{s.t.} \quad & x_1 + x_2 + \cdots + x_t \geq \sum_{l=1}^T d_l, \quad t = 1, \dots, T \\ & x_1, x_2, \dots, x_T \geq 0 \end{aligned}$$

For any period  $t$ , the total quantity produced through period  $t$  is  $\sum_{l=1}^T x_l$  and the total demand through period  $t$  is  $\sum_{l=1}^T d_l$ . The difference of these two quantities is the ending inventory in period  $t$  which must be nonnegative. The constraint matrix for this problem has the property that for every column  $j$ , if  $a_{ij} = 1$  and  $a_{kj} = 1$  for  $k > i$  then  $a_{lj} = 1$  for  $l = i, \dots, k$ . Matrices with this property are called interval matrices.

**Lemma 14.7** Interval matrices are totally unimodular.

**Proof:** Let  $A$  be an interval matrix. Show  $A^\top$  is totally unimodular. Let  $J$  be any collection of columns of  $A^\top$ . The submatrix consisting of the columns in  $A^\top$  indexed by  $J$  is still an interval matrix (now with respect to the rows). Let  $J_1 = \{j \mid j \in J, j \text{ odd}\}$  and  $J_2 = J \setminus J_1$ .  $\square$

We now illustrate a third type of totally unimodular matrix which is a generalization of the node-arc incidence matrix and the interval matrix. Consider the minimum cost network flow problem (*MCNF*). The constraint matrix is a node-arc incidence matrix which is totally unimodular. There are a number

of interesting properties of extreme point solutions to  $(MCNF)$ . The following lemma is left as an exercise. Before proceeding, the reader may wish to review the concepts of tree and spanning tree in Appendix C.

**Lemma 14.8** *If  $A$  is a node-arc incidence matrix of a connected directed graph  $G = (V, A)$ , then  $\text{rank}(A) = m - 1$  where  $m = |V|$ .*

This lemma leads to the following key result.

**Proposition 14.9** *If  $(MCNF)$  is defined on a connected graph  $G = (V, A)$  and  $J \subseteq A$ , then the columns in  $(MCNF)$  indexed by  $J$  are a basis if and only if  $(V, J)$  is a spanning tree of  $G$ .*

**Proof:** First assume the columns indexed by  $J$  correspond to a basic solution. If  $|V| = m$ , by Lemma 14.8  $\text{rank}(A) = m - 1$  and there are  $m - 1$  basic variables in any basic solution so  $|J| = m - 1$ . Since the vertex set,  $V$ , in the directed graph  $G = (V, A)$  has cardinality  $m$  if  $(V, J)$  is not a spanning tree then there is a cycle in  $(V, J)$ . Let  $\{v_1, v_2, \dots, v_{k+1} = v_1\}$  be the vertices in the cycle. Denote by  $B$  the  $(m - 1) \times (m - 1)$  basis matrix whose columns are indexed by  $J$ . If there is a cycle, the basis matrix can be partitioned as

$$B = \begin{bmatrix} B^1 & B^2 \\ 0 & B^3 \end{bmatrix}$$

where the first  $k$  rows of  $B$  correspond to the cycle vertices  $v_1, v_2, \dots, v_k$  and the first  $k$  columns are the basic variables in the cycle. Then adding the  $k$  rows of  $B^1$  gives a 0 vector contradicting the fact that a basis matrix is nonsingular. Thus  $(V, J)$  is a spanning tree. The proof of the proposition in the other direction is left as Exercise 14.2.  $\square$

**Example 14.10** Consider the graph illustrated in Figure 14.1. In this figure the arc flow costs are above each arc and the nonzero supply and demand values are indicated next to each vertex. The arc flow costs are not relevant for the present discussion but are used in the next section on the network simplex

algorithm. The constraint matrix for this transshipment linear program is

$$\left[ \begin{array}{ccccccccc} -1 & -1 & & & & & & & \\ 1 & & -1 & & & & & & \\ & 1 & & & & & & & \\ & & 1 & & -1 & -1 & -1 & & \\ & & & 1 & & 1 & & & \\ & & & & 1 & -1 & & & \\ & & & & & 1 & 1 & & \\ & & & & & & 1 & 1 & \\ & & & & & & & -1 & -1 \end{array} \right] = \left[ \begin{array}{c} x_{12} \\ x_{13} \\ x_{24} \\ x_{45} \\ x_{56} \\ x_{67} \\ x_{78} \\ x_{34} \\ x_{36} \\ x_{37} \\ x_{85} \\ x_{86} \end{array} \right] = \left[ \begin{array}{c} -3 \\ 2 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right]$$

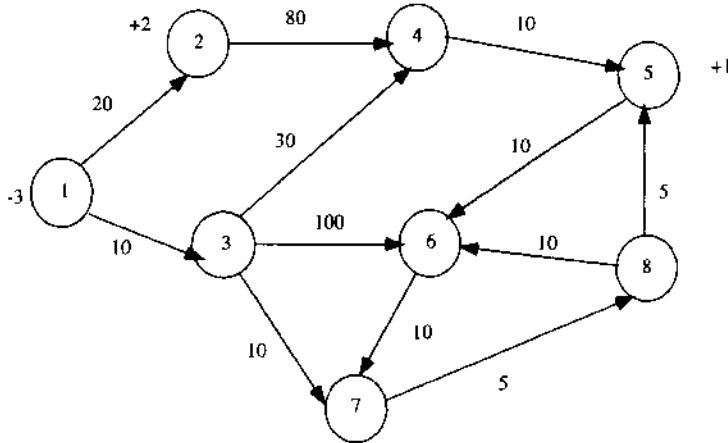
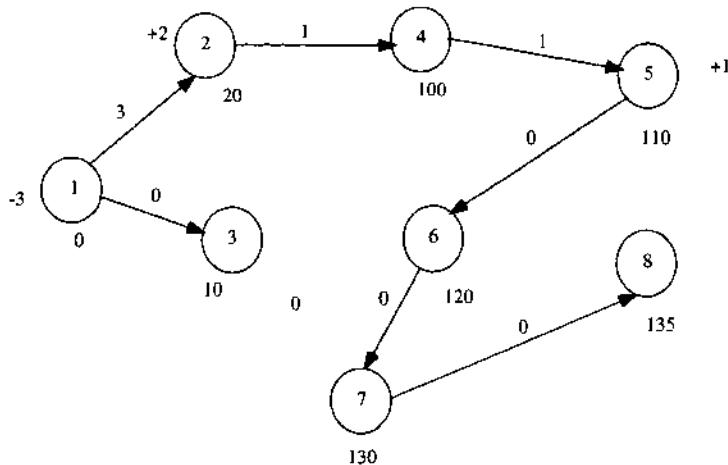
The tree corresponding to a basic feasible solution is given in Figure 14.2. After pivoting on the basic variables the updated matrix is

$$\left[ \begin{array}{cccccc} 1 & & 1 & 1 & 1 & \\ 1 & & -1 & -1 & -1 & \\ 1 & & 1 & 1 & 1 & \\ 1 & & 1 & 1 & 1 & \\ 1 & & 1 & 1 & -1 & \\ 1 & & & 1 & -1 & -1 \\ 1 & & & & -1 & -1 \end{array} \right]$$

The structure of the matrix of the nonbasic variables is particularly interesting. Consider variable  $x_{36}$  for example. Because the basis corresponds to a spanning tree in the underlying graph, there is a unique path from vertex 3 to vertex 6 using only arcs in the spanning tree. The updated column corresponding to variable  $x_{36}$  has a non-zero entry for each arc in this unique path. In particular, if arc  $(v_1, v_2)$  in the spanning tree defining the basis is on the unique path from vertex 3 to 6 in the "forward direction" there is a +1 in row where  $(v_1, v_2)$  is basic. Similarly, if arc  $(v_1, v_2)$  is on the unique path in the "reverse direction" there is a -1 in row where  $(v_1, v_2)$  is basic. See Figure 14.2. (The numbers adjacent to each vertex are dual variable values and used later in the section on the network simplex algorithm.)

This example motivates the following definition. Let  $G = (V, A)$  be a connected directed graph and let  $T = (V, A')$  be a spanning tree on  $V$ . (It is not necessary for  $A' \subseteq A$ .) Define the  $|A'| \times |A|$  matrix  $M$  for  $\alpha = (v_1, v_2) \in A$  and  $\alpha' \in A'$

$$M(\alpha', \alpha) := \begin{cases} +1, & \text{if } \alpha' \text{ is on the unique } v_1, v_2 \text{ path in the forward direction;} \\ -1, & \text{if } \alpha' \text{ is on the unique } v_1, v_2 \text{ path in the reverse direction;} \\ 0, & \text{if } \alpha' \text{ is not on the unique } v_1, v_2 \text{ path.} \end{cases} \quad (14.5)$$

**Figure 14.1** Graph of a Minimum Cost Network Flow Problem**Figure 14.2** Tree Corresponding to a Basic Feasible Solution

Such matrices,  $M$ , are called *network matrices*. We say  $G$  and  $T$  represent the network matrix  $M$ .

**Proposition 14.11** *Let  $G = (V, \mathcal{A})$  be a directed connected graph and let  $T = (V, \mathcal{A}')$  be a directed spanning tree on  $V$ . The corresponding network matrix  $M$  represented by  $G$  and  $T$  is totally unimodular.*

**Proof:** Let  $A$  be the node arc incidence matrix of  $G$  and  $B$  the node arc incidence matrix of  $T$ . Since a spanning tree contains no cycles we can assign positive integers  $i$  to the vertices in  $V$  such that if  $(i, j) \in \mathcal{A}'$  then  $j > i$  and the corresponding node-arc incidence matrix of  $B$  with the first row deleted is upper triangular. Assume row  $i$  of matrices  $A, B$  corresponds to vertex  $i$  of  $V$ . Let  $\tilde{A}$  and  $\tilde{B}$  denote the corresponding matrices with the row 1 deleted. It follows from 14.5 that  $M = \tilde{B}^{-1} \tilde{A}$ . Since  $[\tilde{B}, \tilde{A}]$  are node arc incidence matrices with a row deleted they are totally unimodular. Then by Lemma 14.1,

$$[I, \tilde{B}^{-1} \tilde{A}] = [I, M]$$

is totally unimodular. Then  $M$  is totally unimodular.  $\square$

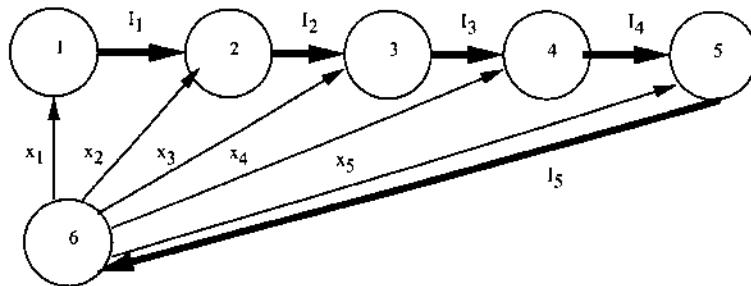
If an arc  $\alpha \in \mathcal{A}$  is added to a spanning tree  $T = (V, \mathcal{A}')$  of a graph  $G = (V, \mathcal{A})$ , a cycle is created. If an arc  $\beta \neq \alpha$  is removed from the cycle a new spanning tree  $T'$  is created. The new spanning tree  $T'$  and original graph  $G$  represent a new network matrix  $M'$ . The new network matrix  $M'$  is actually generated from  $M$  by pivoting on the column in  $M$  corresponding to  $\alpha$ .

**Proposition 14.12** *Let  $G = (V, \mathcal{A})$  be a directed connected graph and  $T = (V, \mathcal{A}')$  be a directed spanning tree on  $V$ . If  $\alpha \in \mathcal{A} \setminus \mathcal{A}'$ ,  $\beta \in \mathcal{A}'$  and  $T' = (V, (\mathcal{A}' \setminus \{\beta\} \cup \{\alpha\}))$  is acyclic then  $M'$  the network matrix defined on  $G' = (V, (\mathcal{A} \setminus \{\alpha\} \cup \{\beta\}))$  is obtained from  $M$  by pivoting on the column corresponding to arc  $\alpha$ .*

**Corollary 14.13** *Network matrices are closed under pivoting.*

Because network matrices are closed under pivoting, we can think of a network matrix as one that is the result of deleting a row from a node-arc incidence matrix and then performing simplex pivots.

**Figure 14.3** Graph of Single Product Lot Sizing Problem



**Example 14.14 (Example 14.6 continued.)** Instead of expressing the demand constraints as  $\sum_{t=1}^T x_t \geq \sum_{t=1}^T d_t$  for  $t = 1, \dots, T$  the more “typical” formulation is  $I_{t-1} + x_t - I_t = d_t$  for  $t = 1, \dots, 5$ . The interesting feature of the second formulation is that the demand constraints are conservation of flow constraints and it is a minimum cost network flow problem. In Figure 14.3 we illustrate the network for a five period model. The inventory variable  $I_t$  corresponds to the arc from node  $t$  to  $t+1$  for  $t = 1, \dots, 5$  and  $x_t$  is the arc from node 6 to node  $t$  for  $t = 1, \dots, 5$ . The constraints for the five period problem based on this graph are

$$\begin{bmatrix} -1 & & & & & \\ & 1 & -1 & & & \\ & & 1 & -1 & & \\ & & & 1 & -1 & \\ & & & & 1 & -1 \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ -\sum_{t=1}^5 d_t \end{bmatrix}.$$

The arcs corresponding to the inventory variables  $I_t, t = 1, \dots, 5$  define a spanning tree in the directed graph for the five period model and are highlighted in bold in the figure. Delete row 1 from the node-arc incidence matrix and pivot on the inventory variables. The resulting matrix corresponding to the

production variables is

$$\begin{bmatrix} -1 \\ -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

and the right hand side is

$$b = \begin{bmatrix} -d_1 \\ -d_1 - d_2 \\ -d_1 - d_2 - d_3 \\ -d_1 - d_2 - d_3 - d_4 \\ -d_1 - d_2 - d_3 - d_4 - d_5 \end{bmatrix}$$

Thus, the network matrix for this example is the negative of the interval matrix introduced in Example 14.6. Reversing the direction of the inventory arcs will give a network matrix identical to the interval matrix.

**Proposition 14.15** *Interval matrices are network matrices.*

We leave it to the reader to prove this proposition in general.

These results and examples suggest several interesting questions. First, is every totally unimodular matrix a network matrix? Although the underlying node-arc incidence matrix for an interval matrix is not immediately obvious, interval matrices are indeed network matrices. Unfortunately, the answer to the question is no. See Bixby [56] for counter-examples. Given that there are totally unimodular matrices that are not network matrices, is it possible to efficiently test if a matrix is a network matrix? Fortunately, the answer to this is yes. See Bixby and Cunningham [57] for a polynomial algorithm and the references contained therein. Finally, given an arbitrary matrix  $A$  with  $\pm 1, 0$  coefficients can one efficiently determine whether or not  $A$  is totally unimodular? The answer is yes, and the polynomial algorithm is based on a very deep theorem due to Seymour [404].

### 14.3 NETWORK SIMPLEX ALGORITHM

In this section we develop the network simplex algorithm for the minimum cost network flow problem (*MCNF*). The network simplex algorithm takes

advantage of the structure of basic feasible solutions for this problem and relies on the properties given in Lemma 14.8 and Propositions 14.9 and 14.12. An immediate consequence of Proposition 14.9 is

**Corollary 14.16** *If  $\bar{x}$  is a basic feasible solution to (MCNF), then the basis matrix  $B$  can be triangularized where the columns correspond to arcs in the basis spanning tree.*

Corollary 14.16 is significant because it implies that the basis matrix is easy to invert. If  $B$  is upper triangular, the  $LU$  decomposition of  $B$  is  $U = B$  and  $L = I$ . There is no fill-in during matrix inversion. Thus, one can expect a well implemented revised simplex algorithm using  $LU$  decomposition, rather than the product form of the inverse, to work well on a linear program with the structure of (MCNF). Refer back to Example 13.9. However, given the special structure of (MCNF) one can do even better and implement the simplex algorithm without floating point operations given integer right hand sides and objective function coefficients.

Given a basic feasible solution, the simplex algorithm must determine if it is optimal, and if it is not (based on a reduced cost calculation), determine a variable to enter the basis and a variable to exit of the basis. The reduced costs are calculated using the dual solution associated with the current basis. Rather than using a BTRAN operation based on the  $LU$  decomposition of the basis matrix, we take advantage of the special structure of the minimum cost network flow linear program. The dual of linear program (MCNF) is

$$\max \sum_{i=1}^m u_i b_i \quad (14.6)$$

$$(MCNF) \quad \text{s.t. } u_j - u_i \geq c_{ij} \text{ for all } (i, j) \in A. \quad (14.7)$$

From complementary slackness, each of the constraints in (14.7) corresponding to a basic variable must hold as a strict equality. Since there are  $m - 1$  basic variables,  $m - 1$  dual constraints (14.7) are tight. Then from Proposition 14.9 the dual constraints corresponding to the arcs in the basis spanning tree are tight. That is, if  $A'$  is the set of basic arcs,

$$u_j - u_i = c_{ij} \quad \text{for all } (i, j) \in A'. \quad (14.8)$$

There are  $m - 1$  equations in (14.8) and  $m$  dual variables so one dual variable is free. Again, using the fact that a spanning tree is acyclic, assume without loss that the vertices are assigned integers  $i = 1, \dots, m$  such that arc  $(i, j)$  in the

basis spanning tree implies  $i < j$ . We are free to arbitrarily set one variable, so let  $u_1 = 0$ . Think of vertex 1 as the *root vertex*. If the root vertex dual variable is known then the dual variables of the vertices adjacent to the root vertex in the basis spanning tree are uniquely determined by (14.8). Repeating this logic gives a dual solution for every vertex in the spanning tree. Then, given the dual solution the reduced cost  $c_{ij} + u_i - u_j$  is calculated for each non basic variable (arc). If an arc has a negative reduced cost then there is a violation of the corresponding dual constraint (14.8) and we must pivot.

There is a nice intuitive way to interpret this dual variable calculation. Think of  $u_j$  as the marginal cost of a product at location  $j$ . If  $c_{ij}$  is the marginal cost of shipping one unit from location  $i$  to location  $j$ , then in an equilibrium condition we would expect  $c_{ij} + u_i - u_j \geq 0$ . If  $c_{ij} + u_i - u_j$  were negative then it would be more profitable to buy our commodity at location  $i$  and then ship it to location  $j$ . Thus, a negative reduced cost implies the system is not in equilibrium so we add arc  $(i, j)$  to the solution and send flow from  $i$  to  $j$ .

**Example 14.17 (Example 14.10 continued)** Consider the spanning tree and associated basic feasible solution of Example 14.10. The objective function vector is (given the matrix order used in Example 14.10)

$$c^T = [ 20 \ 10 \ 80 \ 10 \ 10 \ 10 \ 5 \ 30 \ 100 \ 10 \ 5 \ 10 ].$$

If  $u_1 = 0$ , then  $u_2 = 20$  since  $c_{12} = 20$ . Vertex 4 is adjacent to vertex 2 and  $c_{24} = 80$  so  $u_4 = 100$ . Similarly for the rest of the dual variables. The values of the dual variables for this spanning tree are adjacent to the corresponding vertices in Figure 14.2. The reduced costs for the non basic variables are then

$$\begin{aligned} c_{34} + u_3 - u_4 &= 30 + 10 - 100 = -60 \\ c_{36} + u_3 - u_6 &= 100 + 10 - 120 = -10 \\ c_{37} + u_3 - u_7 &= 10 + 10 - 130 = -110 \\ c_{85} + u_8 - u_5 &= 5 + 135 - 110 = 30 \\ c_{86} + u_8 - u_6 &= 10 + 135 - 120 = 25 \end{aligned}$$

After the dual values and reduced costs are calculated, it is necessary to determine an entering and exit variable if the current solution is not optimal. Select  $x_{34}$  because it is the variable with the most negative reduced cost which does not result in a degenerate pivot. Degeneracy is discussed in the next subsection. If  $x_a$  is selected as the entering variable and arc  $a$  is added to the spanning tree

corresponding to the current basic feasible solution it will create a cycle and it is necessary to pivot out a variable corresponding to one of the arcs in the cycle. See Proposition 14.12. In the simplex algorithm for general linear programs determining the exit variable requires using an FTRAN to update the column of the entering variable and doing a minimum ratio test. Fortunately, in the special case of (MCNF) an FTRAN is not necessary. Because the entering arc has a negative reduced cost we want to send as much flow as possible across that arc in order to reduce the objective function value. In order to maintain conservation of flow it is necessary to adjust the flows on the other arcs in the cycle. Arcs not in the cycle are not affected. This adjustment is made by observing that the new arc  $\hat{a}$  gives an “orientation” to the cycle. We call all arcs in the cycle with the same direction as arc  $\hat{a}$  *forward arcs*, and all arcs in the reverse direction *reverse arcs*. If  $\bar{x}_{ij}$  are current arc flows, conservation of flow is maintained by changing the arc flows in the network as follows:

$$\hat{x}_{ij} = \begin{cases} \bar{x}_{ij} + f, & \text{if } (i, j) \text{ is a forward arc} \\ \bar{x}_{ij} - f, & \text{if } (i, j) \text{ is a reverse arc} \\ \bar{x}_{ij}, & \text{if } (i, j) \text{ is not in the cycle} \end{cases}$$

Since increasing flow along the cycle reduces the objective function value,  $f$  is made as large as possible while keep flow on the reverse arcs nonnegative. That is,

$$f := \min\{\bar{x}_{ij} \mid (i, j) \text{ is a reverse arc}\}. \quad (14.9)$$

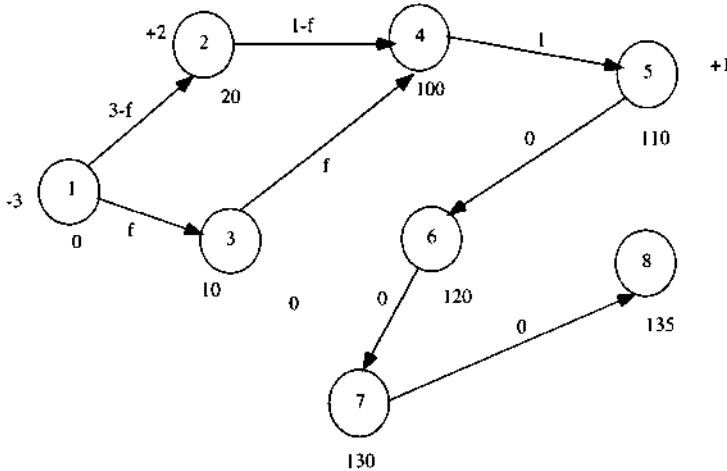
The variable selected to exit the basis is the variable corresponding to the reverse arc with the minimum flow. For now assume ties are broken arbitrarily.

At each pivot, it is not necessary to find the dual variable variables from scratch using (14.8). Assume arc  $\hat{a} = (v_1, v_2)$  is selected to enter the basis with corresponding tree  $T = (V, A')$  and arc  $\bar{a}$  is selected to leave the basis. The set of arcs in  $A' \setminus \{\bar{a}\}$  define two disjoint trees  $T_{v_1}$  and  $T_{v_2}$ , where  $v_1 \in V(T_{v_1})$  and  $v_2 \in V(T_{v_2})$  where  $V(T)$  denotes the vertex set of the arcs in tree  $T$ . It is left as an exercise to show that the new dual variable values are given by

$$\bar{u}_i = \begin{cases} u_i, & i \in V(T_{v_1}) \\ u_i + \bar{c}_{\hat{a}}, & i \in V(T_{v_2}) \end{cases} \quad (14.10)$$

where  $\bar{c}_{\hat{a}}$  is the reduced cost of the entering variable.

**Example 14.18 (Example 14.10 continued)** The nonbasic variable with the most negative reduced cost which does not result in a degenerate pivot is  $x_{34}$ .

**Figure 14.4** Cycle created by adding arc (3,4) to the basis

Adding arc  $(3,4)$  to the spanning tree in Figure 14.2 creates a cycle using vertices  $1, 2, 3$  and  $4$ . See Figure 14.4. There are two reverse arcs,  $(2,4)$  and  $(1,2)$ , in the cycle and from (14.10)

$$f = \min\{\bar{x}_{12}, \bar{x}_{24}\} = \min\{3, 1\} = 1.$$

Thus, variable  $x_{34}$  enters with a value of 1, and variable  $x_{24}$  leaves the basis. The new basic feasible solution with flow values and new dual variable calculation appears in Figure 14.5.

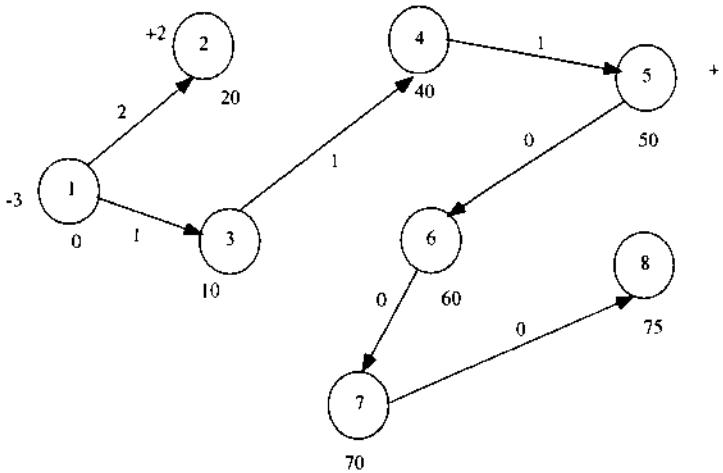
The reduced cost of the entering variable  $x_{34}$  is  $-60$ . Using the dual update formulas (14.10), the vertices  $\{4, 5, 6, 7, 8\}$  in the tree  $T_4$  are decreased by 60. Their new values are

$$u_4 = 40, u_5 = 50, u_6 = 60, u_7 = 70, u_8 = 75.$$

Now calculate the new reduced costs based on the primal-dual solution given in Figure 14.5.

$$c_{24} + u_2 - u_4 = 80 + 20 - 40 = 60$$

**Figure 14.5** Primal and dual solution with arc (3,4) in the basis



$$c_{36} + u_3 - u_6 = 100 + 10 - 60 = 50$$

$$c_{37} + u_3 - u_7 = 10 + 10 - 70 = -50$$

$$c_{85} + u_8 - u_5 = 5 + 75 - 50 = 30$$

$$c_{86} + u_8 - u_6 = 10 + 75 - 60 = 25$$

Pivot variable  $x_{37}$  into the basis since it is the only variable with a negative reduced cost. This introduces the cycle with vertices 3, 4, 5, 6 and 7. See Figure 14.7. Arcs (3,4), (4,5), (5,6) and (6,7) are reverse arcs in the cycle. Since arcs (5,6) and (6,7) are degenerate, there is a tie and more than one variable is eligible to exit the basis. We leave tie breaking strategies to the subsection on degeneracy.

**Algorithm 14.19 (Network Simplex Algorithm)** Simplex implementation for minimum cost network flow problem.

**Step1:** Initialize with a basic feasible solution corresponding to a spanning tree  $T = (V, A')$ .

**Step 2: Dual Variable Calculation:** At the first iteration, initialize the dual variables by  $u_1 \leftarrow 0$  and then solve (14.8). Otherwise, update the dual variables using (14.10).

**Step 3: Reduced Cost Calculation:** Calculate  $c_{ij} + u_i - u_j$  for all nonbasic arcs. If all reduced costs are nonnegative terminate with the optimal solution.

**Step 4: Determine Entering Variable:** Select the variable with the most negative reduced cost and pivot this into the solution thus creating a cycle.

**Step 5: Determine Exit Variable:** Calculate the minimum flow on reverse arcs in the cycle using (14.9). Pivot out variable for which minimum value occurs. Go to dual variable calculation.

Assuming that degeneracy does not cause cycling this process will terminate in a finite number iterations. For the network simplex algorithm, there are some very nice and easy to implement anti-cycling rules. This is the topic of the next subsection. Finally, we recommend that the reader consult the paper by Bradley, Brown, and Graves [71] for an excellent description of the implementation of a large scale network simplex algorithm.

### 14.3.1 Degeneracy and the Network Simplex Algorithm

Minimum cost network flow problems are often highly primal degenerate. In order to guarantee convergence of the network simplex algorithm care must be taken in selecting exiting and entering variables. In the following discussion we assume the graph on which the minimum cost network flow problem is defined is connected.

In calculating the dual variables during network simplex there is a fixed node or root node with a dual value arbitrarily set to zero. Let vertex  $v_0$  be the root vertex. Let  $T_q$  be the spanning tree associated with tableau  $q$ . If arc  $(v_1, v_2)$  is basic, then this arc partitions  $T_q$  into two separate trees,  $T_{q_{v_1}}$  and  $T_{q_{v_2}}$ . If  $v_0 \in V(T_{q_{v_2}})$  then arc  $(v_1, v_2)$  is *directed toward*  $v_0$  and if  $v_0 \in T_{q_{v_1}}$ , then arc  $(v_1, v_2)$  is *directed away* from  $v_0$ . The tree  $T_q$  is *strongly feasible* if every arc in the tree corresponding to a degenerate basic variable is directed away from the root. We leave it as an Exercise 14.16 to show that it is always possible to construct an initial strongly feasible tree. Given a tree solution which is

strongly feasible, we show that it is always possible to pivot so that the next basic feasible solution corresponds to a strongly feasible tree.

Assume at pivot  $q$  variable  $x_a$  is selected as the entering variable and that the corresponding arc is  $a = (v_1, v_2)$ . There is a cycle in the graph  $(V, A_q \cup \{a\})$ . In this graph there is a path from vertex  $v_1$  to vertex  $v_0$  and a path from vertex  $v_2$  to vertex  $v_0$ . These paths must meet, let the *join vertex* be the first vertex where these two paths meet. Conceivably, vertex  $v_0$  is the join vertex.

**Cunningham Exit Rule ([104]):** If, by the minimum ratio test, there is a tie for the exit arc, choose the first candidate in the cycle created by adding arc  $a$  when the cycle is traversed in the direction of  $a$  starting at the join.

**Lemma 14.20** *If the spanning tree  $T_q$  for the basic feasible solution at pivot  $q$  is strongly feasible, and the Cunningham exit variable rule is used, then the spanning tree  $T_{q+1}$  for the basic feasible solution at pivot  $q + 1$  is also strongly feasible.*

In Chapter 5, the Cacioppi and Schrage pivot rules were used to guarantee finite convergence of the simplex algorithm. Essentially, the Cacioppi and Schrage method defines a function on the set of basic variables which assigns a “rank” to each basic variable. The rank assigned to a basic variable  $x_{q_i}$  is  $i$ , its position in the tableau. The exit variable selected, is the variable with maximum rank among the eligible exit variables. The Cunningham exit rule uses exactly the same philosophy. Because the basic feasible solution corresponds to a tree, for each basic variable  $x_{(v_1, v_2)}$  we can define a rank function,  $\sigma((v_1, v_2))$  to be the number of arcs on the path from the root vertex  $v_0$  to vertex  $v_2$ . This is a well defined function since there is a unique path from  $v_0$  to  $v_2$ . Now consider the Cacioppi and Schrage rules altered slightly for the network simplex algorithm.

1. *Initialization:* Initialize each basic variable  $x_a$  to level  $\sigma(a)$ . Initialize all the nonbasic variables to have level  $\infty$ .
2. *Entering Variable:* Select the entering variable from among all level  $\infty$  nonbasic variables with negative reduced cost by any criterion. If there are no level  $\infty$  nonbasic variables with a negative reduced cost, terminate because there is no ray of the degeneracy cone with a negative cost. The validity of this is shown in Lemma 14.22.
3. *Exit Variable:* If there are ties by the minimum ratio test among leaving variables, pivot out the basic variable  $x_a$  with largest value of  $\sigma(a)$ . Because

the trees are strongly feasible, no two exit variable candidates have the same rank. See Lemma 14.21 below. When  $x_a$  is pivoted out of the basis and becomes nonbasic assign it level  $\sigma(a) - 1$  and reassign all variables with level  $\sigma(a)$  or greater, to level  $\infty$ .

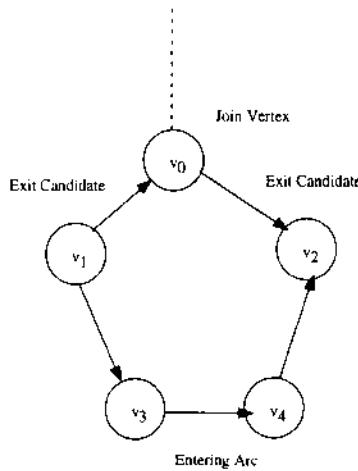
The Cacioppi and Schrage exit rule is equivalent to the Cunningham exit rule. Assume variable  $x_{\hat{a}}$  is selected to enter the basis. Let  $C_{q_{\hat{a}}}$  be the set of *degenerate* arcs in the cycle created by adding arc  $\hat{a}$  to the tree  $T_q$ . If variable  $x_{\hat{a}}$  is the entering variable at pivot  $q$ , the only nonzero elements in the updated column in the degeneracy cone for arc  $\hat{a} = (v_1, v_2)$  correspond to basic variables indexed by arcs in  $C_{q_{\hat{a}}}$ . If  $T_q$  is strongly feasible and  $a \in C_{q_{\hat{a}}}$  is on the path from the join vertex to vertex  $v_1$ , then there is a  $-1$  in the row corresponding to arc  $a$  in the updated column of arc  $\hat{a}$ . If  $T_q$  is strongly feasible and  $a \in C_{q_{\hat{a}}}$  is on the path from the join vertex to vertex  $v_2$ , then there is a  $+1$  in row corresponding to arc  $a$  in the updated column of arc  $\hat{a}$ . See equation (14.5). The Cunningham exit rule selects the first candidate in  $C_{q_{\hat{a}}}$  when the cycle is traversed in the direction of  $\hat{a}$  starting from the join vertex. This corresponds to the candidate nonbasic variable of greatest rank.

**Lemma 14.21** *Assume variable  $x_{\hat{a}}$  is selected to enter the basis. If  $\bar{a}$  is the first arc encountered in  $C_{q_{\hat{a}}}$  when the cycle is traversed in the direction of  $\hat{a}$  starting at the join, then for all  $a \in C_{q_{\hat{a}}}$  such that  $a$  is in the direction of  $\hat{a}$ ,*

$$\sigma(\bar{a}) > \sigma(a).$$

**Proof:** Let  $\hat{a} = (v_1, v_2)$ . The key to the proof is to observe that if  $T_q$  is a strongly feasible tree then any  $a \in C_{q_{\hat{a}}}$  on the path from the join to vertex  $v_1$  is not a candidate to exit the basis because it has a  $-1$  in the row corresponding to basic variable  $x_a$ . Thus, the first arc encountered in  $C_{q_{\hat{a}}}$ , when the cycle is traversed starting at the join in the direction of  $\hat{a}$ , is the last arc in  $C_{q_{\hat{a}}}$  encountered on the path from the root to vertex  $v_2$ . Therefore  $\bar{a}$  strictly maximizes  $\sigma$  over the arcs in  $C_{q_{\hat{a}}}$  which are eligible for exit.  $\square$

By Lemma 14.21, the selection of the exiting arc is well defined. In Figure 14.6 we illustrate why strongly feasible trees are needed. Vertex  $v_0$  is the join vertex. The entering arc is  $\hat{a} = (v_3, v_4)$ . There are two candidate arcs with zero flow to exit the basis  $a_1 = (v_0, v_1)$  and  $a_2 = (v_0, v_2)$ . Both  $a_1$  and  $a_2$  are on the path from  $v_3$  to  $v_4$  in the forward direction so they have a  $+1$  in the updated tableau. Since the join vertex is  $v_0$  both of these arcs have the same rank and the tie breaking rule of using the highest rank arc is no longer valid. The basis illustrated in this figure is not strongly feasible.

**Figure 14.6** Graph of Transshipment Problem

In the case of the minimum cost network flow problem, the Cunningham exit rule alone is sufficient to guarantee finite convergence of the network simplex algorithm. The proof is left as Exercise 14.18. This implies that no entering variable rule is required. However, the Cacioppi and Schrage rule is easy to implement, and one may still wish to use this entering rule for network flow problems because it has the advantage of eliminating variables from consideration. At each pivot it is not necessary to price out a variable with a rank less than  $\infty$ . It has yet to be determined empirically if the extra work of tracking the level of a variable is worth the extra savings in the pricing operation. In the case of the network flow problem we still need to prove the validity of "holding back" the variable with a rank less than  $\infty$ .

The crucial aspect of proving that the Cacioppi and Schrage entering and exit variable strategy is valid is to show that when there are no level  $\infty$  variables with a negative reduced cost, then all the variables with a level of 0 to  $k - 1$  also have a negative reduced cost. It is therefore, valid to conclude there is no extreme ray of the degeneracy cone with negative reduced cost. In the general case, the rank function is given by the matrix order of the basic variables. The key lemma we used in proving the validity of the entering and exit variable rules with this rank function was Lemma 5.18. The crucial idea behind the proof of this lemma was to show that for variable  $x_h$  with rank  $l - 1$ , there was a pivot  $r$ , such that the only nonzero elements of the corresponding column in rows  $l$  or greater were nonnegative, the reduced cost was positive, and no

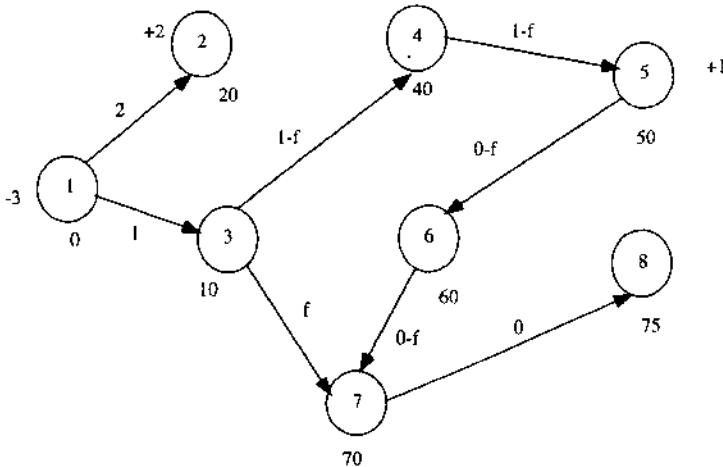
pivots after  $r$  were made in rows 1 through  $l - 1$ . The analog of Lemma 5.18 for the minimum cost network flow problem is Lemma 14.22.

**Lemma 14.22** *If the Cacioppi and Schrage entering and exit variable rules are used for consecutive degenerate pivots of the network simplex algorithm and all nonbasic variables with level  $\infty$  have nonnegative reduced costs, then there is no extreme ray of the degeneracy cone with a negative cost, all variables assigned a rank less than  $\infty$  also have a nonnegative reduced cost and the current tableau is optimal.*

**Proof:** Consider a variable  $x_{\bar{a}}$  which has a level of  $l - 1$  when all level  $\infty$  variables have a nonnegative reduced cost. This implies there was a pivot  $q$ , when variable  $x_{\bar{a}}$  was selected to exit the basis. After pivot  $q$ , there were no pivots on a basic variable with a rank strictly less than  $l$ , that is on a variable  $x_{(s,t)}$  where  $t$  is less than  $l$  arcs from the root vertex. At pivot  $q$  a new variable  $x_{\hat{a}}$  enters the basis. We must show that any nonzero element in the updated column of this variable in a row of the degeneracy cone corresponding to arcs  $l$  or more arcs from the root is -1. Let  $T_q$  be the tree corresponding to the basis at pivot  $q$ . If  $\hat{a} = (v_1, v_2)$ , there is a path in  $T_q$  from vertex  $v_1$  to vertex  $v_2$ . The forward arcs give a +1 in the tableau, the reverse arcs a -1. At pivot  $q$ , the only nonzero elements in column  $\hat{a}$  of the updated degeneracy cone tableau are in rows of basic variables corresponding to elements of  $C_{q\hat{a}}$  (the degenerate arcs in the cycle). This implies for the column corresponding to  $\hat{a}$ , that there are no positive elements in rows of the degeneracy cone with basic variables having rank greater than  $l$ . Then after  $\bar{a}$  is pivoted out of the basis, in tableau  $q + 1$  variable  $x_{\bar{a}}$  has a positive reduced cost and nonnegative elements in rows corresponding to basic variables with rank  $l$  or greater. Also, the Cacioppi and Schrage (or Cunningham) exit rule applied to strongly feasible trees implies that for every degenerate pivot, if a basic variable has rank  $k$  in tableau  $q$  it has rank  $k$  in tableau  $q + 1$  if the exit variable has rank  $k$  or greater. Therefore, if there are no pivots after pivot  $q$  on a variable with rank less than  $l$ , the variables in tableau  $q$  with rank less than  $l$  remain in the basis, are never pivoted out, and their corresponding rows have no effect on the remaining degenerate rows. Then Lemma 5.17 is valid and the result is proved.  $\square$

**Example 14.23 (Example 14.10 continued.)** *After pivoting variable  $x_{34}$  into the basis and reoptimizing the dual variables, variable  $x_{37}$  has the largest negative reduced cost. Adding this variable to the tree results in a cycle with vertices 3, 4, 5, 6, and 7. See Figure 14.7. By the minimum ratio test, arcs (5, 6) and (6, 7) are eligible to exit the basis. Use the Cunningham exit rule. Vertex*

**Figure 14.7** Cycle created by adding arc (3,7) to the basis

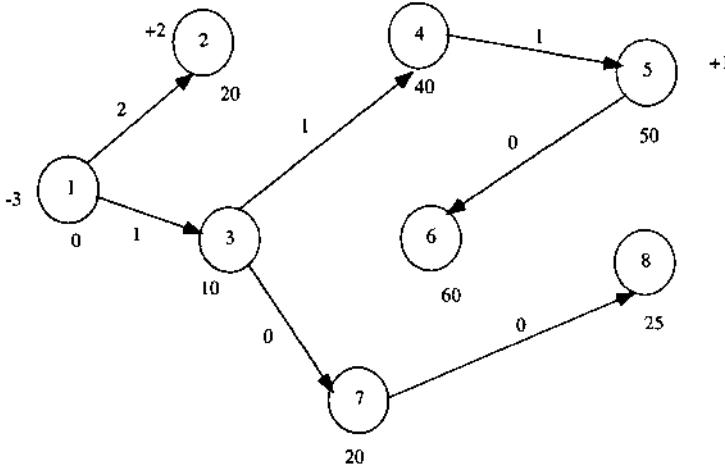


3 is the join vertex. Arc (6,7) is the first arc encountered traversing the cycle in the direction of (3,7) from the join. Therefore, variable  $x_{67}$  is pivoted out of the basis. The new basis and dual solution with variable  $x_{37}$  in the basis is illustrated in Figure 14.8. Notice that the new tree is strongly feasible.

The reduced cost of the entering variable  $x_{37}$  is -50. Using the dual update formulas (14.10), the vertices {7,8} in the tree with arc (7,8) are decreased by 50. Their new values are  $u_7 = 20$ ,  $u_8 = 25$ . It is left as an exercise to complete this example to optimality.

The Cacioppi and Schrage, or Cunningham exit rules are sufficient to guarantee finite convergence of the network simplex algorithm. However, finite convergence does not guarantee that the simplex method will not perform extremely long sequences of degenerate pivots. In fact, even with the Cacioppi and Schrage, or Cunningham pivot rules the network simplex algorithm may perform a sequence of degenerate pivots that is not bounded by a polynomial function in the size of the problem being solved<sup>1</sup>. Cunningham [105] refers

<sup>1</sup>Exponential Growth of the Simplex Method for Shortest Path Problems, unpublished manuscript by J. Edmonds (1970)

**Figure 14.8** Primal and dual solution with arc (3,7) in the basis

to this phenomenon as *stalling*. Cunningham then provides pivot rules which prevent stalling and bound the number of pivots by a low-degree polynomial in the size of the problem. Rules like those proposed by Cunningham have come to be known as *affirmative action* rules. The term affirmative action is used because the basic idea of these rules is that if a variable  $x_i$  has entered the basis more recently than variable  $x_j$ , then the next time both of these variables are eligible to enter the basis, preference is given to variable  $x_j$ .

## 14.4 IMPORTANT NETWORK FLOW PROBLEMS

In this section we look at some important special cases of the minimum cost network flow problem.

### 14.4.1 The Shortest Route Problem

Suppose you are given a road network of the United States and want to find the minimum distance between Lajitas, Texas and Chicago, Illinois. This is an example of the *shortest route problem*. The shortest route problem is a special case of the minimum cost network flow problem where there is a single demand vertex with a demand of one unit and a single supply vertex with a supply of one unit. The cost of using arc  $(i, j)$  is  $c_{ij}$ . The shortest route linear program is

$$\min \quad \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \quad (14.11)$$

$$(SR) \quad \text{s.t.} \quad \sum_{(i,s) \in \mathcal{A}} x_{is} - \sum_{(s,j) \in \mathcal{A}} x_{sj} = -1 \quad (14.12)$$

$$\sum_{(i,k) \in \mathcal{A}} x_{ik} - \sum_{(k,j) \in \mathcal{A}} x_{kj} = 0, \quad k \in V \setminus \{s, t\} \quad (14.13)$$

$$\sum_{(i,t) \in \mathcal{A}} x_{it} - \sum_{(t,j) \in \mathcal{A}} x_{tj} = 1 \quad (14.14)$$

$$x_{ij} \geq 0, \quad (i, j) \in \mathcal{A} \quad (14.15)$$

If there is an optimal solution to linear program  $(SR)$ , then there are no directed cycles with negative cost. If there are no directed cycles with negative cost, the total unimodularity of the constraint matrix implies an integer optimal solution. Since there is a single demand vertex with unit demand there is an optimal solution where arc flows are either 0 or +1. The shortest route from  $s$  to  $t$  is given by the arcs corresponding to an  $x_{ij} = 1$ . If all  $c_{ij} \geq 0$ , and there is at least one directed path from  $s$  to  $t$ , then  $(SR)$  will have an optimal solution. If some arc costs are negative, then the linear program may be unbounded and not give the shortest route. Indeed, if negative costs are allowed then deciding if there is a path of length  $w$  or less is an  $\mathcal{NP}$ -complete problem.

An important related problem is the longest route problem which arises in PERT/CPM applications. See, for example, Schrage [402].

### 14.4.2 The Transportation and Assignment Problem

Many problems in logistics involve shipping a commodity from distribution centers to wholesale or retail outlets at minimum cost. The *transportation*

*problem* is a minimum cost network flow problem defined on a *bipartite graph*. The graph  $G = (V, \mathcal{A})$  is bipartite if

1.  $V = V_1 \cup V_2$ ;
2.  $V_1 \cap V_2 = \emptyset$ ;
3.  $(v_1, v_2) \in \mathcal{A}$  implies  $v_1 \in V_1$  and  $v_2 \in V_2$ .

In the transportation problem, the vertex set  $V$  is partitioned into  $V_1$  and  $V_2$  such that every vertex  $i \in V_1$  is a pure supply vertex and every  $j \in V_2$  is a pure demand vertex. Let  $s_i$ ,  $i \in V_1$  be the supply at vertex  $i$  and  $d_j$ ,  $j \in V_2$  be the demand at vertex  $j$ . The marginal cost of shipping or transporting one unit of product from supply vertex  $i$  to demand vertex  $j$  is  $c_{ij}$ . The transportation linear program (*TLP*) is

$$\min \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \quad (14.16)$$

$$(TLP) \quad \text{s.t.} \quad - \sum_{(i,j) \in \mathcal{A}} x_{ij} = -s_i, \quad i \in V_1 \quad (14.17)$$

$$\sum_{(i,j) \in \mathcal{A}} x_{ij} = d_j, \quad j \in V_2 \quad (14.18)$$

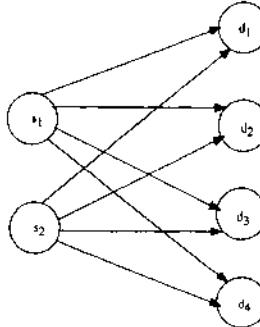
$$x_{ij} \geq 0, \quad (i, j) \in \mathcal{A}. \quad (14.19)$$

This formulation is a slight generalization of model (*TLP*) first given in Subsection 1.3.5 in Chapter 1. An example of a graph underlying a transportation problem with two supply vertices and four demand vertices is given in Figure 14.9.

Like much of linear programming, the history of the transportation problem can be traced to the World War II era. In fact, T.C. Koopmans wrote a paper [272] on the transportation problem based on his war time research as a member of the Combined Shipping Board. See also, Hitchcock [230] and Kantorovich [258] (English version). Although there are real applications which are “pure” transportation problems, and have structure (*TLP*) (see, for example, Bausch et al. [41]), many actual applications have a transportation problem as the underlying structure. A good example is the capacitated plant location problem (*CPL*) introduced in Subsection 1.3.5 in Chapter 1.

A special case of the transportation problem is the *assignment problem*. The assignment problem is a transportation problem where all supplies and demands are equal to 1.

**Figure 14.9** Transportation Problem on Bipartite Graph



### 14.4.3 The Maximum Flow Problem

Consider the problem of sending oil through a network from a source (say Anchorage, Alaska) to a destination or sink (say Chicago, Illinois). The oil is sent through pipelines and these pipelines have capacity on oil throughput. The objective is to maximize the amount of oil sent from the source to the sink. This is known as a *maximum flow problem*. Let  $f$  denote the flow from source  $s$  to sink  $t$ . The maximum flow problem is an example of the minimum cost network flow problem where there are capacities on the arc flow. The capacity on arc  $(i, j)$  is  $c_{ij}$ . The maximum flow linear program is

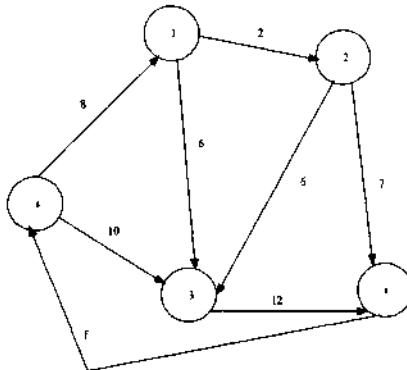
$$\max f \quad (14.20)$$

$$(MF) \quad \text{s.t.} \quad \sum_{(i,s) \in A} x_{is} - \sum_{(s,j) \in A} x_{sj} + f = 0 \quad (14.21)$$

$$\sum_{(i,k) \in A} x_{ik} - \sum_{(k,j) \in A} x_{kj} = 0, \quad k \in V \setminus \{s, t\} \quad (14.22)$$

$$\sum_{(i,t) \in A} x_{it} - \sum_{(t,j) \in A} x_{tj} - f = 0 \quad (14.23)$$

$$0 \leq x_{ij} \leq c_{ij}, \quad (i, j) \in A. \quad (14.24)$$

**Figure 14.10** Network for Max Flow Problem

**Example 14.24** Figure 14.10 illustrates the graph used in this example. The maximum flow problem is to maximize  $f$  subject to the constraints

$$\begin{bmatrix} -1 & -1 & & & & +1 \\ +1 & -1 & -1 & & & \\ & +1 & -1 & -1 & & \\ +1 & +1 & +1 & +1 & -1 & \\ & & & +1 & +1 & -1 \end{bmatrix} \begin{bmatrix} x_{s1} \\ x_{s3} \\ x_{12} \\ x_{13} \\ x_{23} \\ x_{2t} \\ x_{3t} \\ f \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The capacities are

$$c_{s1} = 8, c_{s3} = 10, c_{12} = 2, c_{13} = 6, c_{23} = 6, c_{2t} = 7, c_{3t} = 12.$$

An optimal solution is

$$\bar{x}_{s1} = 4, \bar{x}_{s3} = 10, \bar{x}_{12} = 2, \bar{x}_{13} = 2, \bar{x}_{23} = 0, \bar{x}_{2t} = 2, \bar{x}_{3t} = 12.$$

for a maximum flow of 14.

Consider a seemingly unrelated problem to the maximum flow problem. Think of the graph  $G = (V, A)$  as a network of cities and major highways. The

objective is to launch a military strike against the network so that location  $s$  cannot reinforce location  $t$ . The capacity of each arc represents the cost (perhaps an estimate of casualties) of destroying that arc. Let  $S \subseteq V$  and define

$$\delta(S) := \{(v_1, v_2) \in \mathcal{A} \mid v_1 \in S, v_2 \in V \setminus S, \text{ or } v_2 \in S, v_1 \in V \setminus S\}.$$

The set of arcs in  $\delta(S)$  is called a *cut* or *cutset* of  $G$ . If  $s \in S$  and  $t \in V \setminus S$  we call  $\delta(S)$  an  $s$ - $t$  *cut* or  $s$ - $t$  *cutset*. Clearly, deleting the arcs in an  $s$ - $t$  cut eliminates all paths from  $s$  to  $t$ . The problem of destroying highways at minimum loss in order to prevent city  $s$  from reinforcing city  $t$  reduces to finding a minimum weight cut.

**Example 14.25 (Example 14.24 continued)** Let  $S = \{s\}$ . Then the cut consists of arcs  $(s, 1)$  and  $(s, 3)$ . The cost of destroying these arcs is 18. However, if  $S = \{1, 3\}$  the cut is  $(1, 2)$  and  $(3, t)$  with a weight of 14. This is the value of the minimum cutset which is also the value of the maximum flow. Of course this is not a coincidence!

The dual of the linear program ( $MF$ ) is

$$\min \sum_{(i,j) \in \mathcal{A}} c_{ij} w_{ij} \quad (14.25)$$

$$(DMF) \quad \text{s.t. } u_j - u_i + w_{ij} \geq 0, \quad (i, j) \in \mathcal{A} \quad (14.26)$$

$$u_s - u_t \geq 1 \quad (14.27)$$

$$w_{ij} \geq 0, \quad (i, j) \in \mathcal{A}. \quad (14.28)$$

Given an  $S \subset V$ , with  $s \in S$  and  $t \in V \setminus S$ , let  $\bar{u}_i = 1$ , if  $i \in S$ , 0 otherwise;  $\bar{w}_{ij} = 1$  if  $i \in S$ ,  $j \in V \setminus S$  and 0 otherwise. This is clearly a feasible solution for constraints (14.26)-(14.28). Thus, for every  $s$ - $t$  cutset there is a feasible dual solution. However, through duality we obtain the following stronger classic result (see Ford and Fulkerson [152]) known as the max flow-min cut theorem.

**Theorem 14.26** *If there is an optimal solution to a maximum flow linear program then the value of the maximum flow is equal to the value of the minimum  $s$ - $t$  cutset.*

**Proof:** Clearly any  $s$ - $t$  cutset is a feasible dual solution. Show that there is an optimal dual solution that is also an  $s$ - $t$  cutset. Then by strong duality the maximum flow value is equal to the minimum  $s$ - $t$  cutset value.

To see that there is an optimal dual solution which is an  $s-t$  cutset observe that constraint (14.23) in the linear program ( $MF$ ) is redundant. Then there is an optimal dual solution in which  $\bar{u}_t = 0$ . The constraint matrix for the linear program ( $MF$ ) is a node-arc incidence matrix with simple upper and lower bound constraints, and hence is totally unimodular. The dual constraint matrix is also totally unimodular so there is an optimal dual solution which is integer. Since the dual objective function is a minimization it follows that when an optimal dual solution exists, there is always an optimal dual solution  $\bar{u}$  that is binary. Let  $S = \{i \mid \bar{u}_i = 1\}$ . In order to satisfy (14.27) it must be true that  $s \in S$  so  $\delta(S)$  is an  $s-t$  cutset. The capacity of the cutset  $\delta(S)$  is the optimal value of the dual problem and the theorem is proved.  $\square$

Although there are applications, see for example Chalmet, Francis, and Saunders [86], which are pure maximum flow problems, the maximum flow problem is often solved as a subproblem for more complicated problems. Earlier we studied both the primal simplex algorithm and dual simplex algorithm. Another algorithm which has been successfully applied to minimum cost network flow problems is the primal-dual algorithm. At each iteration of the primal-dual algorithm a maximum flow problem is solved in an attempt to satisfy primal feasibility. The maximum flow problem is also used as a subproblem to find violated cuts for integer programming problems. See, for example, Van Roy and Wolsey [438], Cunningham [106], Padberg and Wolsey [367], and Grötschel and Holland [213]. See Ahuja, Magnanti, and Orlin [7] for other applications of the max flow problem.

**Example 14.27** Consider the traveling salesman problem on the directed graph  $G = (V, A)$ . Assume  $s \in V$  is the home city. One formulation of the tour breaking constraints due to Dantzig, Fulkerson, and Johnson [115] is

$$\sum_{\substack{i \in V \setminus S \\ j \in S}} x_{ij} \geq 1, \quad S \subseteq V \setminus \{s\}. \quad (14.29)$$

The number of tour breaking constraints in (14.29) is an exponential function of  $|V|$  so it is desirable to add only those that are not redundant. This requires solving a *separation problem* which takes an  $\bar{x}_{ij}$  as input and decides if  $\bar{x}_{ij}$  satisfies (14.29), and if not, gives at least one violated constraint. The use of separation algorithms is studied in greater detail later in Section 15.3 in Chapter 15. The separation problem can be solved by setting up a max flow problem for each  $t \in V$  and  $t \neq s$ . Let  $y_{ij}^t$  be the flow on arc  $(i, j)$ .

$$\max f_t \quad (14.30)$$

$$(SMF_t) \quad \text{s.t.} \quad \sum_{(i,s) \in \mathcal{A}} y_{is}^t - \sum_{(s,j) \in \mathcal{A}} y_{sj}^t + f_t = 0 \quad (14.31)$$

$$\sum_{(i,k) \in \mathcal{A}} y_{ik}^t - \sum_{(k,j) \in \mathcal{A}} y_{kj}^t = 0, \quad k \in V \setminus \{s, t\} \quad (14.32)$$

$$\sum_{(i,t) \in \mathcal{A}} y_{it}^t - \sum_{(t,j) \in \mathcal{A}} y_{tj}^t - f_t = 0 \quad (14.33)$$

$$0 \leq y_{ij}^t \leq \bar{x}_{ij}, \quad (i, j) \in \mathcal{A} \quad (14.34)$$

If the optimal solution value of  $(SMF_t)$  is at least 1, then none of the subtour breaking constraints in (14.29) are violated. However, if there is a  $t \in V$  such that the max flow value of  $(SMF_t)$  is strictly less than one then a constraint in (14.29) is violated. Let  $\bar{u}_k^t$  be the associated optimal dual variable values on the constraints (14.32). Define  $S = \{i \in V \mid \bar{u}_k^t = 1\}$ . Then this  $S$  provides a violated constraint (14.29).

#### 14.4.4 Complexity

In this section we have detailed some important special cases of the minimum cost network flow problem. Since these problems are special cases of the minimum cost network flow problem, the network simplex algorithm can be used to solve these problems. For the general minimum cost network flow problem the network simplex algorithm is not a polynomial algorithm. However, strongly polynomial algorithms do exist for each of the special cases considered, the shortest route, transportation, assignment, and max flow problem. See Ahuja, Magnanti and Orlin [7] for an excellent survey of these algorithms. If one of these specialized network flow problems needs to be solved many times, perhaps as a subroutine in a more complicated problem, then the specialized algorithm is probably the way to go. However, the network simplex algorithm has the advantage of solving a broad class of problems and has proven extremely effective in solving large scale network flow problems.

Orlin [360] has shown that if the variable with minimum reduced cost is used to select the entering variable and a lexicographic rule used to prevent cycling then the maximum number of pivots for the minimum cost network flow problem when applied to the assignment or shortest path problem on  $G = (V, \mathcal{A})$  is  $O(|V|^2|\mathcal{A}| \log_2(|V|))$ . Goldfarb and Hao [188] give a polynomial time implementation of a primal simplex algorithm for the maximum flow problem. Orlin [359] and Plotkin and Tardos [421] have polynomial time implementations of dual network simplex algorithms.

Because network simplex algorithms are so fast, it is beneficial to know if a constraint system  $Ax = b$  can be transformed into a network flow problem using variable scaling and pivot operations. Bixby and Cunningham [57] give a polynomial algorithm for determining whether this can be done.

## 14.5 ALMOST NETWORK PROBLEMS

### 14.5.1 Generalized Networks

A natural extension of the minimum cost network flow problem is to allow for flows with *gains* or *losses*. In the minimum cost network flow problem on  $G = (V, \mathcal{A})$ , for each  $(i, j) \in \mathcal{A}$  one unit of flow enters vertex  $j$  on arc  $(i, j)$  for every unit of flow that leaves vertex  $i$ . However, in many applications this may not be the case. For example, in cash management applications money invested at time  $t$  will return the principal plus interest at time  $t + 1$ . If there is a vertex for each time period  $t$  the arc from  $t$  to  $t + 1$  will have a multiplier of  $1 + r$  where  $r$  is an interest rate. Stated differently, if one unit of flow (a dollar) leaves vertex  $t$  for vertex  $t + 1$  then the flow into vertex  $t + 1$  is  $1 + r$ . If gains and losses are allowed it is no longer the case that basic feasible solutions correspond to spanning trees. However, there are codes which have customized the revised simplex algorithm for networks with gains and losses. See, for example, Glover et al. [186].

### 14.5.2 Leontief Substitution Systems

The matrix  $A$  is *pre-Leontief* if there is at most one positive element in every column. If there is exactly one positive element per column and there is an  $\bar{x} \geq 0$  such that  $A\bar{x} > 0$  then  $A$  is a *Leontief* matrix. If  $b \geq 0$  and  $A$  is a Leontief (pre-Leontief) matrix, then  $Ax = b$ ,  $x \geq 0$  is a *Leontief (pre-Leontief) substitution system*. See Veinott [445] and Koehler, Whinston, and Wright [268] for a thorough discussion of Leontief substitution systems.

In the previous subsection, generalized networks were developed as an extension of network flow problems by allowing the negative element in each column of the vertex-arc incidence matrix to be an arbitrary negative real number instead of -1. A pre-Leontief matrix further extends the vertex-arc incidence matrix of generalized networks by allowing more than one negative element in a column. The corresponding graph structure is called a *Leontief directed*

*hypergraph*  $G = (V, \mathcal{H})$  on vertices in  $V$  and hyperarcs in  $\mathcal{H}$ . Each hyperarc is an ordered pair  $(J, k)$ , where  $J \subseteq V$ ,  $k \in V \setminus J$ . Vertex  $k$  is the *head* of the hyperarc and  $J$  is the *tail set*. We allow *tailless or source* hyperarcs  $(\emptyset, k)$  directed into  $k$ , and *headless or sink* hyperarcs  $(J, \emptyset)$  directed out of tail set  $J$ . The coefficient associated with the head of a hyperarc is +1. *Tail weights*  $\{a_j[J, k] \mid j \in J\}$  give the magnitudes of corresponding negative coefficients at tails  $j \in J$ . Models can have several “parallel” hyperarcs with different tail weights connecting the same  $(J, k)$  pair. In the interest of notational simplicity, we assume each  $(J, k)$  pair identifies a unique hyperarc. Parallel hyperarcs are easily made unique through the introduction of artificial vertices.

A *flow* on Leontief directed hypergraph  $G = (V, \mathcal{H})$  is a set of hyperarc values  $\{x[J, k] \mid (J, k) \in \mathcal{H}\}$  conforming to conservation and nonnegativity constraints.

$$\sum_{(J,k) \in \mathcal{H}} x[J, k] - \sum_{\substack{(J,l) \in \mathcal{H} \\ k \in J}} a_k[J, l]x[J, l] = b_k, \quad k \in V \quad (14.35)$$

$$x[J, k] \geq 0, \quad (J, k) \in \mathcal{H} \quad (14.36)$$

Right hand sides  $b_k$  denote the net demands at vertices  $k$ .

Leontief substitution systems arise in linear programming models of multi-echelon production systems. These systems have the feature that production of an item, e.g. a lawn mower, will have many inputs and production of this item creates demand for components, e.g. wheels, pistons etc. See, for example, Martin, Rardin, and Campbell [317] and Jeroslow et al. [243] for examples of these multi-echelon models and other examples of Leontief substitution systems. Leontief substitution systems are also used later in Chapter 16. When all of the tail weights are integer, Jeroslow et al. [243] give a strongly polynomial solution algorithm. In the case of two nonzero elements per column Adler and Cosares [1] give a strongly polynomial solution algorithm.

### 14.5.3 Multicommodity Flows

In the formulation of the minimum cost network flow problem, there was a single commodity and  $x_{ij}$  represented the flow of this commodity on arc  $(i, j)$ . However, in many important applications there are multiple commodities. For example, a large corporation in developing a minimum cost distribution plan will have many distinct commodities that must be shipped from plants and distribution centers to retail centers. Unfortunately, in most real applications the

different commodities share common resources such as truck and distribution center capacity so a separate minimum cost network flow problem cannot be solved for each commodity. The generic formulation for the multicommodity minimum cost network flow problem on a graph  $G = (V, \mathcal{A})$  with commodities  $l = 1, \dots, m$  is

$$\min \quad \sum_{l=1}^m \sum_{(i,j) \in \mathcal{A}} c_{ij}^l x_{ij}^l \quad (14.37)$$

$$\text{s.t.} \quad \sum_{(i,k) \in \mathcal{A}} x_{ik}^l - \sum_{(k,j) \in \mathcal{A}} x_{kj}^l = b_k^l, \quad k \in V, \quad l = 1, \dots, m \quad (14.38)$$

$$\sum_{l=1}^m x_{ij}^l \leq d_{ij}, \quad (i,j) \in \mathcal{A} \quad (14.39)$$

$$x_{ij} \geq 0, \quad (i,j) \in \mathcal{A} \quad (14.40)$$

In this formulation,  $x_{ij}^l$  is the flow of commodity  $l$  on arc  $(i,j)$ ,  $c_{ij}^l$  is the marginal cost of commodity  $l$  on arc  $(i,j)$ ,  $b_k^l$  is the net demand for commodity  $l$  at vertex  $k$ , and  $d_{ij}$  is the capacity of arc  $(i,j)$ . Unlike the minimum cost network flow model, the constraint matrix of the multicommodity minimum cost network flow model is not totally unimodular. Thus, vertex solutions are not naturally integer. This is unfortunate because it prevents development of a network simplex algorithm which does not require explicit inversion of the basis. Several specialized results do exist. For example, Evans [144] shows how to transform the multicommodity transportation problem with at most two sources or sinks into a single commodity capacitated transportation problem.

## 14.6 INTEGER POLYHEDRA

It is of interest to know when a polyhedron is an integer polyhedron because this is a sufficient condition for the associated linear programming relaxation to have integer extreme points. When the constraint matrix is totally unimodular, an integer right hand side vector guarantees an integer polyhedron. This is a sufficient, but not necessary condition, and in this section we look at other sufficient conditions. Throughout this discussion we assume that  $A$  is a rational matrix.

This discussion is from Giles and Pulleyblank [181] and Schrijver [403]. Recall that a polyhedron is an integer polyhedron if and only if every minimal face contains an integer vector. If a polyhedron is pointed, then it is an integer

polyhedron if and only if every extreme point is an integer vector. An equivalent definition is provided in the following proposition.

**Proposition 14.28** *Let  $P = \{x \mid Ax \geq b\}$  with  $A$  and  $b$  rational. Then  $P$  is an integer polyhedron if and only if the optimal solution value of  $\min\{c^\top x \mid Ax \geq b\}$  is an integer for every integer vector  $c$  for which there is an optimal solution.*

**Proof:** First assume  $P$  is an integer polyhedron. If there is an optimal solution value to the linear program  $\min\{c^\top x \mid Ax \geq b\}$ , then there is a face (a minimal face)  $F$ , of  $P$  such that every  $x \in F$  is an optimal solution to this linear program. By definition of integer polyhedron, there is an integer vector  $\bar{x}$  in  $F$ . If  $c$  is an integer vector then  $c^\top \bar{x}$  is an integer.

Now assume that for every integer vector  $c$ , for which  $\min\{c^\top x \mid Ax \geq b\}$  exists, the optimal solution value is an integer. Show that this implies  $P$  is an integer polyhedron. Prove the contrapositive and assume that  $P$  is not an integer polyhedron. Then there is a face  $F$  of  $P$ , which does not contain an integer vector. We can take  $F$  to be a minimal face so there is an index set  $I(F)$  such that  $F = \{x \mid a^i x = b_i, i \in I(F)\}$  where the  $a^i$  are rows of  $A$ . Refer to condition 1 of Proposition A.6 in Appendix A. Since  $A$  and  $b$  are rational we can take  $a^i$  and  $b_i$  for  $i \in I(F)$  to be integers. If  $F$  does not contain an integer vector then by Proposition 4.10 there are  $\bar{u}_i, i \in I(F)$  such that  $\sum_{i \in I(F)} \bar{u}_i a^i = c$  where  $c$  is integer, but  $\sum_{i \in I(F)} \bar{u}_i b_i$  is not an integer. Adding positive integers to the  $u_i$  does not change the result so we can assume  $\bar{u}_i \geq 0$ . Define  $\bar{u}_i = 0$  for all  $i \notin I(F)$ . Then for any  $\bar{x} \in F$ ,  $(\bar{x}, \bar{u})$  is a complementary primal-dual feasible solution to the linear program  $\min\{c^\top x \mid Ax \geq b\}$ . Therefore,  $(\bar{x}, \bar{u})$  is an optimal solution to this linear program and by strong duality  $\sum_{i \in I(F)} \bar{u}_i b_i$  is the optimal solution value. This is not an integer and the contrapositive is proved.  $\square$

The rational system  $Ax \geq b$  is *totally dual integer* (TDI) if and only if there is an optimal integer dual solution  $u$  to the system

$$\min\{c^\top x \mid Ax \geq b\} = \{u^\top b \mid u^\top A = c^\top, u \geq 0\}$$

for every integer  $c$  when the optimum exists. The following corollary is immediate from Proposition 14.28.

**Corollary 14.29** *If  $Ax \geq b$  is a TDI system and  $b$  is integer, then  $\{x \mid Ax \geq b\}$  is an integer polyhedron.*

At first, it may appear that Corollary 14.29 implies that if  $Ax \geq b$  is TDI, then the polyhedron  $\{x \mid Ax \geq b\}$  is integer since we can always scale the rational system  $Ax \geq b$  so that  $b$  is integer. The problem with this logic is that if  $Ax \geq b$  is TDI it does not follow that  $\alpha Ax \geq \alpha b$ , where  $\alpha$  is an integer, is TDI. We leave it to the reader to construct an example where the TDI property is destroyed by multiplication of an integer. This suggests that TDI is very much an algebraic property.

**Example 14.30** Total dual integrality is not a geometric property of polyhedrons, but rather an algebraic or representation property. Consider the linear program

$$\begin{aligned} \min \quad & -x_1 - 2x_2 \\ \text{s.t.} \quad & x_1 - x_2 \geq 0 \\ & -x_1 - x_2 \geq -2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

The extreme points of this polyhedron (actually a polytope) are

$$\{(0, 0), (2, 0), (1, 1)\}$$

so it is an integer polyhedron. However, the representation given is not TDI. For the given objective function vector  $(-1, -2)$  the tight constraints for the optimal solution of  $(1, 1)$  are  $x_1 - x_2 = 0$  and  $-x_1 - x_2 = 2$ . But  $(-1, -2)$  is not an integer combination of  $(1, -1)$  and  $(-1, -1)$ . Indeed, the optimal dual variables for these constraints are 0.5 and 1.5, respectively.

Example 14.30 illustrates that it is possible to have an algebraic description of an integer polyhedron that is not TDI. However, the algebraic description of a polyhedron is not unique. We show now that every integer polyhedron does have a TDI description. The reader may wish to review the concept of a Hilbert basis in Section 4.6 in Chapter 4 before proceeding.

**Proposition 14.31** The rational system  $Ax \geq b$  is TDI if and only if for each minimal face  $F$  of the polyhedron  $P = \{x \mid Ax \geq b\}$ , the rows of  $A$  which are active form a Hilbert basis for the cone which they generate.

**Proof:** First assume  $Ax \geq b$  is TDI. Let  $F \neq \emptyset$  be an arbitrary minimal face of  $P$ . By Proposition A.6 in Appendix A, there is an index set  $I(F)$  such that

$F = \{x \mid a^i x = b_i, i \in I(F)\}$  where  $a^i$  are rows of  $A$ . Let  $c$  be an arbitrary integer vector in  $\text{cone}\{a^i \mid i \in I(F)\}$ . This implies there is a  $\bar{u} \geq 0$  such that  $\bar{u}^\top A = c^\top$  with  $\bar{u}_i = 0$ , for all  $i \notin I(F)$ . Then every  $x \in F$  is an optimal solution to the linear program  $\min\{c^\top x \mid Ax \geq b\}$  because  $\bar{u}$  is a complementary dual feasible solution. By the TDI hypothesis we assume  $\bar{u}$  is integer. Since  $c$  was an arbitrary integer vector in the cone, the rows of  $A$  indexed by  $I(F)$  define a Hilbert basis.

Now assume the active rows of  $A$  which define an arbitrary minimal face form a Hilbert basis for the cone which they generate. Let  $c$  be an arbitrary integer vector such that  $\min\{c^\top x \mid Ax \geq b\}$  exists. Index the tight rows associated with an optimal primal solution  $\bar{x}$  which define a face  $F$  by  $I(F)$ . Since these rows define a Hilbert basis there is an integer  $\bar{u}$  with  $u_i = 0$  for all  $i \notin I(F)$ . The pair  $(\bar{x}, \bar{u})$  are a complementary and feasible primal-dual pair and hence optimal. Since  $c$  was arbitrary,  $Ax \geq b$  is TDI.  $\square$

If the constraints defining each face of a polyhedron are a Hilbert basis for the cone they generate then the system is TDI. This suggests taking a representation that is not TDI and adding redundant constraints until the constraints defining every face form a Hilbert basis.

**Proposition 14.32** *If  $P = \{x \mid Ax \geq b\}$  is an integer polyhedron with  $A, b$  rational, then there exists a TDI system  $A'x \geq b'$  with  $b'$  integer such that  $P = \{x \mid A'x \geq b'\}$ .*

**Proof:** Since  $A, b$  are rational, we can rescale so that all data is integer so assume  $b$  is an integer vector. Let  $F$  be an arbitrary minimal face of  $P$ . Then there is an index set  $I(F)$  of rows from  $A$  such that  $F = \{x \mid a^i x = b_i, i \in I(F)\}$ . Let  $K(F)$  be the cone generated by the rows of  $A$  which define  $F$ . By Proposition 4.23 there exist integer vectors  $d^i, i \in H$  such that the  $a^i$  define a Hilbert basis for  $K(F)$ . Then if  $c \in K(F)$  is an integer vector,  $c$  can be written as a nonnegative integer combination of the  $d^i, i \in H$ . Also, since the  $d^i \in K(F)$ , they are also a nonnegative combination of the rows of  $F$ . This implies the linear program  $\min\{(d^i)^\top x \mid Ax \geq b\}$  has an optimal solution  $d_{0i}$ . Since  $P$  is an integer polyhedron, by Proposition 14.28,  $d_{0i}$  is an integer. Then  $(d^i)^\top x \geq d_{0i}$  is a redundant constraint for  $P$  and can be added without affecting  $P$ . Adding the constraints  $(d^i)^\top x \geq d_{0i}, i \in H$  for each minimal face gives a description such that the constraints of every minimal face define a Hilbert basis. Then by Proposition 14.31 this gives a TDI representation with an integer right hand side.  $\square$

The following corollary is an immediate from this proposition.

**Corollary 14.33** *The rational polyhedron  $P$  is an integer polyhedron if and only if there is a TDI system  $Ax \geq b$  with integer  $b$  such that  $P = \{x \mid Ax \geq b\}$ .*

Cook, Lovász, and Schrijver [99] give a polynomial algorithm in fixed dimension for testing if a system is TDI. Also see the paper by Cook [98] for operations that preserve the TDI structure.

**Example 14.34 (Matroid Polyhedra)** *Let  $\mathcal{F}$  be a collection of subsets of a finite ground set  $E$ . For each  $S \subseteq E$  define the rank function  $r(S)$  by*

$$r(S) = \max\{|I| \mid I \subseteq S, I \in \mathcal{F}\}. \quad (14.41)$$

*Then  $I \in \mathcal{F}$  if and only if  $r(I) = |I|$ . An ordered pair  $M := (E, \mathcal{F})$  is a matroid if*

1.  $J \subset I \in \mathcal{F}$  implies  $J \in \mathcal{F}$ ;
2. For any  $S \subseteq E$ , all maximal (with respect to inclusion) members of  $\mathcal{F}$  which are subsets of  $S$  have the same rank.

Whitney [448] is considered the classic on matroid theory. Given a matroid  $M = (E, \mathcal{F})$  and cost vector  $c$  in  $\mathbb{R}^{|E|}$ , define the linear program

$$\max \sum_{e \in E} c_e x_e \quad (14.42)$$

$$(MLP) \quad \text{s.t.} \quad \sum_{e \in S} x_e \leq r(S), \quad S \subseteq E \quad (14.43)$$

$$x_e \geq 0, \quad e \in E. \quad (14.44)$$

It is left as Exercise 14.21 to construct an example matroid where the constraint matrix of  $(MLP)$  is not totally unimodular. However, the underlying polyhedron of  $(MLP)$  is an integer polyhedron.

The dual linear program  $(DMLP)$  is

$$\min \sum_{S \subseteq E} r(S) u_S \quad (14.45)$$

$$(FMLP) \quad \text{s.t.} \quad \sum_{\{S \mid e \in S\}} u_S \geq c_e, \quad S \subseteq E \quad (14.46)$$

$$u_S \geq 0, \quad S \subseteq E. \quad (14.47)$$

Given  $c \in \mathbb{R}^{|E|}$ , sort the elements of  $E$  such that  $c_{e_1} \geq c_{e_2} \geq \dots \geq c_{e_{|E|}}$ . Let  $k$  be the last index  $i$  for which  $c_{e_i} > 0$ . (If  $c_{e_1} \leq 0$  then the optimal primal-dual solution is  $x = 0$ ,  $u = 0$ .) Define  $S_i = \{e_1, e_2, \dots, e_i\}$ . Now construct the following primal-dual solution

$$\bar{x}_{e_i} = \begin{cases} 1, & \text{if } i = 1; \\ r(S_i) - r(S_{i-1}), & \text{if } i = 2, \dots, k; \\ 0, & \text{if } i > k; \end{cases} \quad (14.48)$$

$$\bar{u}_S = \begin{cases} c_{e_i} - c_{e_{i+1}}, & \text{if } S = S_i, 1 \leq i \leq k-1; \\ c_{e_k}, & \text{if } S = S_k; \\ 0, & \text{otherwise.} \end{cases} \quad (14.49)$$

We leave as an exercise to verify that the primal-dual solution given in (14.48)-(14.49) is primal-dual optimal. Given the optimality of (14.48)-(14.49), if  $c$  is an integer vector, it follows that  $\bar{u}$  is also an integer vector. Since the rank function has the set of nonnegative integers as its range, it follows from Corollary 14.29 that the matroid polyhedron of  $(MLP)$  is an integer polyhedron. This proof is due to Edmonds [137].

Example 14.34 is an application of linear programming and polyhedral theory to a combinatorial optimization problem. The merging of polyhedral and linear programming theory with combinatorial optimization began in earnest in the 1950s. The general discipline is now known as *polyhedral combinatorics*. Two pioneers of this fascinating field are J. Edmonds and D.R. Fulkerson. See Pulleyblank [374] for a comprehensive survey of major results in this area.

Another approach to proving that a polyhedron is integer is to use projection or inverse projection. In the following discussion we assume that the polyhedron of the primal and dual problem is pointed so that minimal faces are extreme points. By Proposition 2.31, applying projection to a primal problem generates all the extreme points of the dual polyhedron. If these extreme points are integer whenever  $c$  is an integer vector, then the primal system is TDI. Similarly, one can apply inverse projection to the primal and generate all the extreme points of the primal. From Theorem 3.3 applying inverse projection to the primal linear program

$$(LP) \quad \begin{aligned} & \min c^\top x \\ & \text{s.t. } Ax = b \\ & \quad x \geq 0 \end{aligned}$$

gives a new linear program

$$\begin{aligned} \min \quad & c^T T z \\ \text{s.t.} \quad & \sum_{i=1}^q d_i z_i = 1 \\ & z_i \geq 0, \quad i = 1, \dots, r \\ & x - T z = 0. \end{aligned}$$

The following proposition provides an easy way to test if the polyhedron in  $(LP)$  is an integer polyhedron.

**Proposition 14.35** *If*

$$\begin{aligned} P &= \{x \mid Ax = b, x \geq 0\} \\ &= \{x \mid x - Tz = 0, \sum_{i=1}^q d_i z_i = 1, d_i > 0, i = 1, \dots, q, z_i \geq 0, i = 1, \dots, r\} \end{aligned}$$

and  $(1/d_i)Te^i$  is an integer vector for  $i = 1, \dots, q$ , then  $P$  is an integer polyhedron.

**Proof:** This follows directly from Theorem 3.3. If  $\bar{x}$  is an extreme point of  $P$ , then there is an index  $k \leq q$  such that  $\bar{x} = (1/d_k)Te^k$ . By hypothesis,  $(1/d_k)Te^k$  is integer, so all extreme points of  $P$  are integer.  $\square$

Proposition 14.35 is a sufficient, but not necessary condition for an integer polyhedron. As discussed in Chapter 3, some of the  $z_i$  may not correspond to extreme point solutions, thus  $(1/d_i)Te^i$  may not be an integer vector even when  $P$  is an integer polyhedron. Proposition 14.35 provides an easy integrality proof for Leontief substitution systems with integer data.

**Corollary 14.36** *If  $A$  is an integer pre-Leontief matrix, every positive element of  $A$  is +1, and  $b$  is an integer nonnegative vector then  $P = \{x \mid Ax = b, x \geq 0\}$  is an integer polyhedron.*

**Proof:** Apply inverse projection to the system

$$Ax - x_0 b = 0, \quad x_0 = 1, \quad x, x_0 \geq 0. \quad (14.50)$$

Project out the first row of the system  $Ax - x_0 b = 0$ . Without loss, assume  $b_1 > 0$ . Let  $a_{1j} = 1$  for all  $j \in J^+$  and  $a_{1j} < 0$  for all  $J^-$ . Project out the first row using the variable substitution

$$x_j = - \sum_{l \in J^-} a_{1l} u_{jl} - u_{j0} b_0, \quad j \in J^+ \quad (14.51)$$

$$x_j = \sum_{l \in J^+} u_{lj}, \quad j \in J^- \quad (14.52)$$

$$x_0 = \sum_{l \in J^+} u_{l0}. \quad (14.53)$$

The resulting system is

$$A'u = 0, \quad \sum_{l \in J^+} u_{l0} = 1, \quad u \geq 0. \quad (14.54)$$

Consider column  $j$  of  $A$ . If  $j \in I_+$  then every coefficient on variable  $j$  is non-positive in rows 2 through  $m$  of  $A$  since  $A$  is pre-Leontief. Since  $a_{1l} < 0$  for  $l \in J^-$ , the substitution in (14.51) leads to variables with negative integer coefficients. If  $j \in I^-$  there is at most one row where a +1 appears. Then making the substitution does not result in any column with more than one +1. Thus, the substitution (14.51)-(14.53) results in the  $A'$  matrix in (14.54) also being pre-Leontief. Also, the coefficients in the non-homogeneous constraint are all +1. After projecting out all the rows, the resulting system is

$$\sum_{i=1}^q z_i = 1, \quad z_i \geq 0, \quad i = 1, \dots, r.$$

The original variables are related to the  $z$  variables by  $x = Tz$  and the integrality of  $A$  along with (14.51)-(14.53) imply  $T$  is an integer matrix. Then  $x^i = Te^i$  is integer for  $i = 1, \dots, q$  and by Proposition 14.35,  $P$  is an integer polyhedron.  $\square$

We conclude the chapter by investigating conditions under which applying projection to a TDI system results in a lower dimensional TDI system. Cook [98] has shown the following

**Lemma 14.37** *Total dual integrality is maintained under projection of a variable  $y$ , if it occurs in each constraint with coefficient 0 or  $\pm 1$ . That is if the system*

$$a_i x + y \geq b_i \quad i = 1, \dots, m' \quad (14.55)$$

$$a_i x - y \geq b_i \quad i = m' + 1, \dots, m'' \quad (14.56)$$

$$a_i x \geq b_i \quad i = m'' + 1, \dots, m \quad (14.57)$$

is TDI, the system

$$(a_i + a_j)x \geq b_i + b_j \quad i = 1, \dots, m'; j = m' + 1, \dots, m'' \quad (14.58)$$

$$a_i x \geq b_i \quad i = m'' + 1, \dots, m \quad (14.59)$$

is also TDI.

The proof of Lemma 14.37 is left as an exercise. This result is extended in the following proposition.

**Proposition 14.38** *Assume that the system  $Ay + Bx \geq b$  is TDI and that  $A$  is a totally unimodular matrix. Then projecting out the  $y$  variables gives a system that is also TDI.*

**Proof:** By Lemma 14.37, projecting out a single  $y_i$  variable results in a TDI system. By Lemma 14.2 the resulting coefficient matrix on the  $y$  is still totally unimodular so this process can be repeated until all the  $y$  variables are projected out and the resulting system is TDI.  $\square$

Proposition 14.38 is used by Martin [316] in an integrality proof of a graph optimization problem (the minimum spanning tree problem).

## 14.7 CONCLUSION

Many important applications give rise to linear programming models where all, or a large proportion, of the constraints have a network flow structure. The constraint matrix of a network flow linear program has a structure that enables these linear programs to be optimized very quickly. In addition, the associated polyhedron is integer which means that no enumeration is required in order to find integer solutions. Thus, not only can the linear programming relaxation of a network problem be solved quickly, no enumeration is necessary. The reader wishing to learn more about network flow linear programs should consult the classic book, "Flows in Networks," by Ford and Fulkerson [152]. The reader interested in more recent books with all or considerable emphasis on network flow models and algorithms should consult *Network Flows Theory, Algorithms, and Applications* by Ahuja, Magnanti, and Orlin [7] and *Linear Programming and Network Flows* by Bazaraa, Jarvis, and Sherali [43]. For

emphasis on combinatorial optimization we recommend *Combinatorial Optimization Networks and Matroids* by Lawler [280], *Geometric Algorithms and Combinatorial Optimization* by Grötschel, Lovász, and Schrijver [212], and *Combinatorial Optimization Algorithms and Complexity* by Papadimitriou and Steiglitz [368]. The paper by Giles and Pellyblank [181] is a particularly lucid introduction to total dual integrality.

## 14.8 EXERCISES

- 14.1 Prove that if  $A$  is a node-arc incidence matrix of the connected directed graph  $G = (V, \mathcal{A})$  then  $\text{rank}(A) = m - 1$  where  $m = |V|$ . This is Lemma 14.8.
- 14.2 Prove that if  $(MCNF)$  is defined on a connected graph  $G = (V, \mathcal{A})$  and  $J \subset \mathcal{A}$ , such that  $(V, J)$  is a spanning tree of  $G$ , then the columns in  $(MCNF)$  indexed by  $J$  are a basis.
- 14.3 Prove Corollary 14.16.
- 14.4 Complete Example 14.10 to optimality.
- 14.5 Explain how to carry out the simplex network algorithm when the variables have simple upper and lower bounds.
- 14.6 Explain why it is legitimate to assume that  $\det(\tilde{B}) = \pm 2$  in the proof of Lemma 14.2.
- 14.7 Prove Lemma 14.1.
- 14.8 Show that if  $A$  is a matrix with nonzero elements of  $\pm 1$ , and no square submatrix of  $A$  has a determinant of  $\pm 2$ , then pivoting on a nonzero element will not create a submatrix with determinant  $\pm 2$ .
- 14.9 In the proof of Proposition 14.11 we used the fact that  $M = \tilde{B}^{-1}\tilde{A}$ . Verify that this is indeed true.
- 14.10 Prove Proposition 14.12.
- 14.11 Prove Proposition 14.15.
- 14.12 Construct an example where  $Ax \geq b$  is TDI,  $\alpha$  is an integer, but  $\alpha Ax \geq \alpha b$  is not TDI.

- 14.13 Prove that for any rational system  $Ax \geq b$ , there is a positive rational number  $\alpha$  such that  $\alpha Ax \geq \alpha b$  is TDI.
- 14.14 Show that the transportation linear program (*TLP*) will have a feasible solution if  $\sum_{i=1}^n s_i \geq \sum_{j=1}^m d_j$ .
- 14.15 The proof of Proposition 14.36 is based on inverse projection. Provide a proof of this proposition by applying projection to the dual problem.
- 14.16 Given  $G = (V, A)$  show how to construct an initial strongly feasible tree (possibly through the addition of artificial arcs).
- 14.17 Prove Lemma 14.20.
- 14.18 Prove that the network simplex algorithm with the Cunningham exit rule is finite. Recall that in calculating the dual variables at each pivot we are free to arbitrarily fix the value of the dual variable at the root node. If the dual value for the root node is fixed to zero at every iteration, then if every basis is strongly feasible, the sum of the dual variables is strictly decreasing at each iteration. This implies simplex cannot cycle.
- 14.19 Prove that at each iteration of the network simplex algorithm, the new dual solution is given by (14.10).
- 14.20 Show that if there exist upper bounds on the variables, every system  $Ax = b$ ,  $x \geq 0$  can be converted into a system  $\hat{A}\hat{x} = \hat{b}$ ,  $\hat{x} \geq 0$  where  $\hat{A}$  is a pre-Leontief matrix. Note that it may not be the case that  $\hat{b} \geq 0$ . This result is due to Rardin [375].
- 14.21 Give an example of a matroid such that the constraint matrix (14.43)-(14.44) of the linear program (*MLP*) is not totally unimodular.
- 14.22 Verify that the primal-dual solution of Example 14.34 given in (14.48)-(14.49) is primal-dual optimal for the linear program (*MLP*).

## LARGE INTEGER PROGRAMS: PREPROCESSING AND CUTTING PLANES

### 15.1 FORMULATION PRINCIPLES AND TECHNIQUES

All integer linear programming models are not created equal. Linear programming based branch-and-bound is the most commonly used algorithm for optimizing integer programming models. If the linear programming relaxation of the integer program is naturally integer, then no enumeration is needed. Most interesting integer programming models are not naturally integer, but *reformulating or tightening* the model so that the convex hull of integer solutions more closely approximates the linear programming relaxation polyhedron will greatly reduce the amount of enumeration. Williams [451, 456] was among the first to stress the importance of correctly formulating an integer program. Consider the following examples.

**Example 15.1** Recall the simple plant location problem first introduced in Sub-section 1.3.5 in Chapter 1 as the lock box location problem. See also, Section 10.3 in Chapter 10.

The parameters are  $m$ , the number of customer locations to be assigned a plant;  $n$ , the number of potential plant locations;  $c_{ij}$ , the annual cost associated with serving customer  $j$  from plant  $i$ ; and  $f_i$ , the annual fixed cost of operating a plant at location  $i$ . The variables are  $x_{ij}$ , a binary variable which is equal to 1 if customer  $j$  is assigned to plant  $i$  and 0 if not;  $y_i$ , a binary variable which is equal to 1 if the plant at location  $i$  is open and 0 if not.

The simple plant location integer program is

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} + \sum_{i=1}^m f_i y_i \\ (SPL) \quad \text{s.t.} \quad & \sum_{i=1}^n x_{ij} = 1, \quad j = 1, \dots, m \\ & x_{ij} \leq y_i, \quad i = 1, \dots, n, \quad j = 1, \dots, m \\ & x_{ij}, y_i \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \end{aligned} \quad (15.1)$$

$$(15.2)$$

$$(15.3)$$

An equivalent formulation with exactly the same set of integer solutions is

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} + \sum_{i=1}^m f_i y_i \\ (SPLAGG) \quad \text{s.t.} \quad & \sum_{i=1}^n x_{ij} = 1, \quad j = 1, \dots, m \\ & \sum_{j=1}^m x_{ij} \leq m y_i, \quad i = 1, \dots, n \\ & x_{ij}, y_i \in \{0, 1\}, \quad i = 1, \dots, n, \quad j = 1, \dots, m. \end{aligned} \quad (15.4)$$

$$(15.5)$$

$$(15.6)$$

The difference in the two formulation is the way the setup forcing constraints are defined. In model (SPL) there are  $n \times m$  constraints of the form  $x_{ij} \leq y_i$ , which force  $y_i$  to one if plant  $i$  is serving any customer  $j$ . In model (SPLAGG), the forcing constraints have been aggregated into  $n$  constraints of the form  $\sum_{j=1}^m x_{ij} \leq m y_i$ . It is still the case that if plant  $i$  serves any customer  $j$  then plant  $i$  is forced to be open. The set of feasible solutions for the model is the same, but model (SPLAGG) has far fewer constraints than model (SPL). If there were 50 potential plant sites and 400 customers model (SPL) has 20000 forcing constraints but model (SPLAGG) has only 50 forcing constraints! Below are the formulations and solutions for the linear programming relaxations of models (SPL) and (SPLAGG) for the three plant, five customer test case of Example 10.6 in Chapter 10. The formulation and solution for (SPL) are

```

MIN 2 X11 + 3 X12 + 4 X13 + 5 X14 + 7 X15 + 2 Y1 + 4X21
+ 3 X22 + X23 + 2 X24 + 6 X25 + 3 Y2 + 5 X31 + 4 X32 +
2 X33 + X34 + 3 X35 + 3 Y3
SUBJECT TO
2)   X11 + X21 + X31 =      1
3)   X12 + X22 + X32 =      1

```

```

4) X13 + X23 + X33 = 1
5) X14 + X24 + X34 = 1
6) X15 + X25 + X35 = 1
7) X11 - Y1 <= 0
8) X12 - Y1 <= 0
9) X13 - Y1 <= 0
10) X14 - Y1 <= 0
11) X15 - Y1 <= 0
12) X21 - Y2 <= 0
13) X22 - Y2 <= 0
14) X23 - Y2 <= 0
15) X24 - Y2 <= 0
16) X25 - Y2 <= 0
17) X31 - Y3 <= 0
18) X32 - Y3 <= 0
19) X33 - Y3 <= 0
20) X34 - Y3 <= 0
21) X35 - Y3 <= 0
END

```

## OBJECTIVE FUNCTION VALUE

1) 16.00000

VARIABLE	VALUE	REDUCED COST
X11	1.000000	0.000000
X12	1.000000	0.000000
Y1	1.000000	0.000000
X33	1.000000	0.000000
X34	1.000000	0.000000
X35	1.000000	0.000000
Y3	1.000000	0.000000

NO. ITERATIONS= 17

The linear programming relaxation solution value for model (SPL) is 16 and the solution is integer. Solving this problem with LINDO requires no branches since the linear programming relaxation is integer and takes 17 total pivots. The formulation and solution for the aggregated model (SPLAGG) are

MIN 2 X11 + 3 X12 + 4 X13 + 5 X14 + 7 X15 + 2 Y1 + 4X21

```
+ 3 X22 + X23 + 2 X24 + 6 X25 + 3 Y2 + 5 X31 + 4 X32 +
2 X33 + X34 + 3 X35 + 3 Y3
```

SUBJECT TO

- 2) X11 + X21 + X31 = 1
- 3) X12 + X22 + X32 = 1
- 4) X13 + X23 + X33 = 1
- 5) X14 + X24 + X34 = 1
- 6) X15 + X25 + X35 = 1
- 7) X11 + X12 + X13 + X14 + X15 - 5 Y1 <= 0
- 8) X21 + X22 + X23 + X24 + X25 - 5 Y2 <= 0
- 9) X31 + X32 + X33 + X34 + X35 - 5 Y3 <= 0

END

LP OPTIMUM FOUND AT STEP 3  
OBJECTIVE VALUE = 12.600004

NEW INTEGER SOLUTION OF 16.000000  
BOUND ON OPTIMUM: 16.00000  
ENUMERATION COMPLETE. BRANCHES= 3 PIVOTS= 28

LAST INTEGER SOLUTION IS THE BEST FOUND  
RE-INSTALLING BEST SOLUTION...

: nonzero

OBJECTIVE FUNCTION VALUE

1) 16.00000

VARIABLE	VALUE	REDUCED COST
Y1	1.000000	2.000000
Y3	1.000000	3.000000
X11	1.000000	0.000000
X12	1.000000	0.000000
X33	1.000000	0.000000
X34	1.000000	0.000000
X35	1.000000	0.000000

The linear programming relaxation solution value for model (SPLAGG) is 12.6 and is fractional. LINDO<sup>1</sup> requires 3 branches and a total of 28 pivots to find

---

<sup>1</sup>LINDO has built-in preprocessing implementing many of the ideas developed in this chapter. We have turned off the preprocessing in running this example.

and prove integer optimality. Considerably more work is required for the smaller formulation! Although the polyhedron for the linear programming relaxation of (SPL) is not an integer polyhedron, in many cases the linear programming relaxation is naturally integer! If an  $x_{ij} = 1$  in the linear programming relaxation, then the forcing constraint  $x_{ij} \leq y_i$  forces  $y_i = 1$ . In the aggregated model, the linear programming relaxation value of  $y_i$  is  $\bar{y}_i = (\sum_{j=1}^m \bar{x}_{ij}/m)$ . Unless plant  $i$  servers every customer, its relaxation value will be fractional. Model (SPL) is superior to model (SPLAGG). More constraints help! There is considerable empirical evidence which indicates that it is easier to solve a larger integer program, which requires minimal enumeration, than an equivalent smaller integer program, which has a weak linear programming relaxation and requires a lot of enumeration.

**Example 15.2** This example illustrates the geometry of the simple plant location formulations. Let

$$\Gamma = \{(x_1, x_2, y) \mid x_1 \leq y, x_2 \leq y, x_1, x_2, y \in \{0, 1\}\}.$$

The constraints  $x_1 \leq y$  and  $x_2 \leq y$  can be aggregated into the single constraint  $x_1 + x_2 \leq y$  giving the equivalent formulation

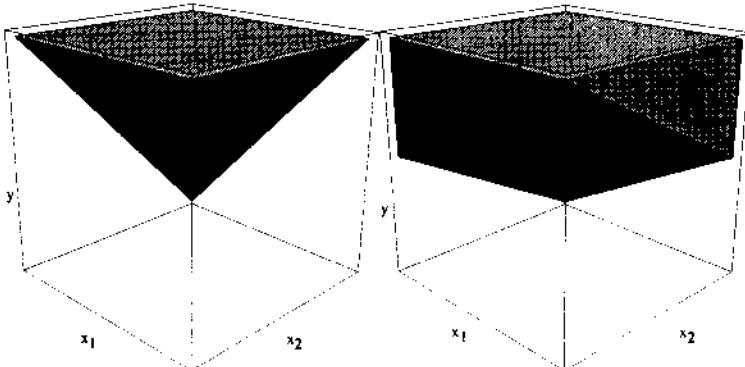
$$\Gamma' = \{(x_1, x_2, y) \mid x_1 + x_2 \leq 2y, x_1, x_2, y \in \{0, 1\}\}.$$

The sets  $\Gamma$  and  $\Gamma'$  are equivalent because they define exactly the same set of binary solutions. The set of binary solutions is

$$(0, 0, 0), (0, 0, 1), (0, 1, 1), (1, 0, 1), (1, 1, 1)$$

In Figure 15.1 we have the polytopes  $\bar{\Gamma}, \bar{\Gamma}'$ , representing the linear relaxations of  $\Gamma$  and  $\Gamma'$ , respectively. Compare  $\bar{\Gamma}$  on the left, with  $\bar{\Gamma}'$  on the right. From the Figure we can see that  $\text{conv}(\Gamma) = \bar{\Gamma} \subset \bar{\Gamma}'$ . Replacing the single constraint  $x_1 + x_2 \leq 2y$  with two constraints  $x_1 \leq y$  and  $x_2 \leq y$  is very desirable if a linear programming based branch-and-bound algorithm is used.

**Example 15.3** The multiproduct dynamic lot size model was introduced in Subsection 1.3.4 in Chapter 1. The parameters are  $T$ , the number of time periods;  $N$ , the number of products;  $d_{it}$ , the demand for product  $i$  in period  $t$ ;  $f_{it}$ , the fixed cost associated with production of product  $i$  in period  $t$ ;  $h_{it}$ , the marginal cost of holding one unit of product  $i$  in inventory at the end of period  $t$ ;  $c_{it}$ , the marginal production cost of one unit of product  $i$  in period  $t$ ; and  $g_t$ , the production capacity available in period  $t$ . The variables are  $x_{it}$ , the units of

**Figure 15.1** Polyhedra of Tight and Loose Formulations

product  $i$  produced in period  $t$ ;  $I_{it}$ , the units of product  $i$  held in inventory at the end of period  $t$ ; and  $y_{it}$ , a binary variable which is fixed to 1 if there is nonzero production of product  $i$  in period  $t$ , otherwise it is fixed to 0. The multiproduct dynamic lot size mixed integer program is

$$\begin{aligned}
 \min \quad & \sum_{i=1}^N \sum_{t=1}^T (c_{it}x_{it} + h_{it}I_{it} + f_{it}y_{it}) \\
 (MPDLS) \quad \text{s.t.} \quad & \sum_{i=1}^N x_{it} \leq g_t, \quad t = 1, \dots, T \\
 & I_{i,t-1} + x_{it} - I_{it} = d_{it}, \quad i = 1, \dots, N, \quad t = 1, \dots, T \\
 & x_{it} - M_{it}y_{it} \leq 0, \quad i = 1, \dots, N, \quad t = 1, \dots, T \\
 & x_{it}, I_{it} \geq 0, \quad i = 1, \dots, N, \quad t = 1, \dots, T \\
 & y_{it} \in \{0, 1\}, \quad i = 1, \dots, N, \quad t = 1, \dots, T.
 \end{aligned}$$

In this formulation, the constraints  $x_{it} \leq M_{it}y_{it}$  are fixed charge constraints which force  $y_{it}$  to be one if  $x_{it}$  is nonzero thus capturing the fixed cost associated with any nonzero production. The "M" coefficient on  $y_{it}$  is often called "big M" in the literature since  $M$  must be big enough to guarantee that no feasible production schedules are eliminated. To illustrate, consider the data for the two product, five time period model in Example 12.20 in Chapter 12. Since the capacity in each period is 200 units, a big  $M$  of 1000 units is valid. Using these data with a big  $M$  of 1000 results in a linear programming relaxation solution value of 5845.6. All positive integer variables in the solution are fractional and finding an optimal integer solution requires 12 branches and 77 pivots.

Because the machine capacity in each period is 200 units, it is valid to reduce big  $M$  to 200. In fact, since production never exceeds demand it is valid to replace the big  $M$ , for product  $i$ , in each time period  $t$ , with the minimum of 200 (production capacity) and  $\sum_{l=t}^T d_{il}$ . The new linear programming relaxation solution for this big  $M$  is given is 6353.000. The optimal integer solution value is 6495 so there has been a huge decrease in the integrality gap. With the reduced big  $M$ , LINDO requires six branches and 52 pivots. Both the number of branches and pivots is reduced by a smaller value of big  $M$ . As the size of the formulation increases the effect of big  $M$  becomes more crucial. See Camm, Rauri, and Tsubakitani [80] for computational results showing the dramatic effect of reducing big  $M$  for lot sizing problems.

These examples demonstrate that the integer linear programming formulation has a big impact on the solution effectiveness of a linear programming based branch-and-bound algorithm. The more naturally integer (i.e. tight) the formulation, the better. Formulating “good” integer linear programs is the topic of this chapter. The generic (*MIP*) model is  $\min \{c^T x \mid Ax \geq b, x \in \Gamma\}$  where  $\Gamma = \{x \in \Re^n \mid Bx \geq d, x \geq 0, x_j \in \mathbb{Z}, j \in I\}$ . We assume that there is some special structure on  $\Gamma$ . Ideally, we want  $\bar{\Gamma}$  to approximate  $\text{conv}(\Gamma)$  as closely as possible. In Section 15.2 we discuss methods for preprocessing problems so that  $\bar{\Gamma}$  more closely approximates  $\text{conv}(\Gamma)$ . In Section 15.3 we discuss methods for generating valid inequalities which can be added to the formulations so that  $\bar{\Gamma}$  more closely approximates  $\text{conv}(\Gamma)$ . Again, the whole philosophy is make the formulation as tight as possible and more amenable to solution by linear programming based branch-and-bound. In Section 15.4 we continue with this theme and discuss branch-and-cut where the philosophy is to try to tighten the model *after* entering the branching phase. Using branch-and-cut, valid cuts are generated after a subset of the variables are fixed to zero and one. In Section 15.5, we describe *lifting* methods which give coefficients on the variables fixed at one or zero so that the cut is valid for the entire search tree. In Section 15.6, we describe Lagrangian cuts, which tie together the methods of this chapter with those in Chapter 12. In Section 15.7 we mention some of the commercial software packages for solving mixed integer programming problems and describe several test banks of integer programming problems which have played an important role in empirical work. Concluding remarks are in Section 15.8. Exercises are provided in Section 15.9.

## 15.2 PREPROCESSING

Loosely speaking, by preprocessing we refer to what is done to a formulation to make it more amenable to solution *before* solving the linear programming relaxation. There are two objectives of preprocessing. The first objective is to make the linear programming relaxation of the mixed integer linear program

$$(MIP) \quad \begin{aligned} & \min c^T x \\ & \text{s.t. } Ax = b \\ & \quad x \in \Gamma \end{aligned}$$

easy to solve. This might involve eliminating redundant constraints, fixing variables, scaling coefficients, improving bounds on variables, etc. The second objective is to make the linear programming relaxation of  $(MIP)$  as tight as possible. Preprocessing has a tremendous effect on the solvability of a linear or integer linear program. Brarley, Mitra, and Williams [74] were among the first to preprocess linear programs prior to applying a simplex solver. For a more recent study, see Andersen and Andersen [11]. See also Bixby [55], Gondzio [195], and Lustig, Marsten, and Shanno [302], for the effect of preprocessing on barrier algorithm solutions of linear programs and Hoffman and Padberg [235] and Savelsbergh [398] for computational studies on the effect of preprocessing integer programs. We now describe a number of methods, taken from these papers, designed to achieve both goals.

Assume the canonical form of a constraint is

$$\sum_{j \in I^+} a_j x_j + \sum_{j \in C^+} a_j x_j - \sum_{j \in I^-} a_j x_j - \sum_{j \in C^-} x_j \geq b \quad (15.7)$$

where the  $x_j$  are continuous for all  $j \in C^+ \cup C^-$ , integer for all  $j \in I^+ \cup I^-$  and all  $a_j$  are positive. For all variables  $x_j$ , denote by  $l_j$  the lower bound (possibly  $-\infty$ ) and  $h_j$  the upper bound (possibly  $\infty$ ).

### 15.2.1 Rounding

If  $C^+ = C^- = \emptyset$ ,  $a_j$  is integer for all  $j \in I^+ \cup I^-$  and  $\alpha = \gcd(a_j \mid a_j \in I^+ \cup I^-)$  then the conical form of the constraint is

$$\sum_{j \in I^+} a_j x_j - \sum_{I^-} a_j x_j \geq b \quad (15.8)$$

which is equivalent to

$$\sum_{j \in I^+} (a_j / \alpha) x_j - \sum_{I^-} (a_j / \alpha) x_j \geq b / \alpha. \quad (15.9)$$

The set of integer solutions to (15.9) is identical to the set of integer solutions to

$$\sum_{j \in I^+} (a_j/\alpha)x_j - \sum_{I^-} (a_j/\alpha)x_j \geq \lceil b/\alpha \rceil. \quad (15.10)$$

However, when  $b/\alpha$  is not an integer, (15.10) admits fewer continuous solutions than (15.9) and will help reduce the integrality gap. The constraint matrix coefficients in a (MIP) model will not always be integer. Integer coefficients are needed for this rounding process. Hoffman and Padberg [235] discuss a method of preprocessing pure integer programs so that all coefficients are integer. They call the rounding method developed in this subsection Euclidean rounding.

### 15.2.2 Coefficient Reduction

Coefficient reduction is a very simple, but powerful idea, used throughout this chapter. In Section 15.3 we show how to generate cutting planes using coefficient reduction. For now, we make the simple observation that if  $C^- = I^- = \emptyset$  and  $a_j \geq 0$  for all  $j \in I^+ \cup C^+$ , then it is valid to reduce the coefficients on the integer variables to  $b$ . That is, (15.7) is equivalent to

$$\sum_{j \in I^+} \min\{a_j, b\}x_j + \sum_{j \in C^+} a_jx_j \geq b. \quad (15.11)$$

By equivalent, we mean that (15.7) and (15.11) have the same set of nonnegative solutions with  $x_j$  integer for all  $j \in I^+$ . Additionally, if  $C^-$  or  $I^-$  is not empty then upper bounds on the variables in these sets are used as follows. Define

$$\lambda := b + \sum_{j \in C^-} a_j h_j + \sum_{j \in I^-} a_j h_j. \quad (15.12)$$

Then (15.7) is equivalent to

$$\sum_{j \in I^+} \min\{a_j, \lambda\}x_j + \sum_{j \in C^+} a_jx_j - \sum_{j \in I^-} a_jx_j - \sum_{j \in C^-} a_jx_j \geq b. \quad (15.13)$$

Clearly, the smaller the value of  $\lambda$  in (15.13) the better the linear programming relaxation. This means that deriving upper bounds which are as small as possible is important. This is the topic of the next subsection.

### 15.2.3 Tightening Bounds

Improving the upper and lower bounds on variables will, in general, improve the performance of linear programming algorithms and improve the value of the linear programming relaxation. One simple way to improve upper and lower bounds on individual variables is to sequentially consider each constraint (15.7), and for each  $k \in C^+$  observe that (15.7) implies

$$a_k x_k \geq b - \sum_{\substack{j \in C^+ \\ j \neq k}} a_j x_j - \sum_{j \in I^+} a_j x_j + \sum_{j \in C^-} a_j x_j + \sum_{j \in I^-} a_j x_j. \quad (15.14)$$

The smallest the right hand side of (15.14) can be is

$$b - \sum_{\substack{j \in C^+ \\ j \neq k}} a_j h_j - \sum_{j \in I^+} a_j h_j + \sum_{j \in C^-} a_j l_j + \sum_{j \in I^-} a_j l_j. \quad (15.15)$$

Therefore, it is valid to reset  $l_k$  to

$$(b - \sum_{\substack{j \in C^+ \\ j \neq k}} a_j h_j - \sum_{j \in I^+} a_j h_j + \sum_{j \in C^-} a_j l_j + \sum_{j \in I^-} a_j l_j)/a_k \quad (15.16)$$

if this results in a larger value than the user input value of the lower bound. The lower bounds for the variables in  $I^+$  are adjusted in a similar fashion. Any fractional lower bound on an integer variable may be rounded up to the closest integer value. Using similar logic the upper bounds of variables indexed by  $C^- \cup I^-$  are adjusted by

$$(\sum_{j \in C^+} a_j h_j + \sum_{j \in I^+} a_j h_j - \sum_{\substack{j \in C^- \\ j \neq k}} a_j l_j - \sum_{j \in I^-} a_j l_j - b)/a_k \quad (15.17)$$

if this results in a new smaller upper bound on variable  $x_k$ . This method of adjusting upper and lower bounds is repeated until there is no improvement in an upper or lower bound. Once the lower and upper bounds are calculated one can apply coefficient reduction on the integer variable coefficients. In the next example we illustrate the result of the **TITAN** command in LINDO which implements both bound tightening and coefficient reduction.

**Example 15.4** [Example 15.3 continued.] Consider the two product, five time period dynamic lot sizing example with a big  $M$  of 1000 for each fixed charge variable. Initially, there is a lower bound of 0 (the nonnegativity constraints) on every variable and an upper bound of  $\infty$  on the continuous variables and 1 on the 0/1 fixed charge variables. The capacity constraints have the structure  $\sum_{i=1}^2 x_{it} \leq 200$  for  $t = 1, \dots, 5$ , or in canonical form  $-\sum_{i=1}^2 x_{it} \geq -200$ .

Then from (15.17) there is an upper bound of 200 on all of the  $x_{it}$  production variables.

Next, process the demand constraints. The demand constraints are equality constraints and imply for each product and each period,  $t = 1, \dots, 4$  a canonical form of  $-I_{i,t-1} - x_{it} + I_{it} \geq -d_{it}$  and in period five,  $-I_{i,4} - x_{i5} \geq -d_{i5}$ . Starting in period 5, and working back to period 1 using (15.17) implies the upper bound of  $x_{it}$  is  $\min\{200, \sum_{k=t}^5 d_{ik}\}$ . The upper bounds on the  $I_{it}$  variables are left as Exercise 15.9. The upper bounds on the  $x_{it}$  variables are used to reduce the values of big  $M$  in the fixed charge constraints using the coefficient reduction given by (15.13). To give the reader some feeling for just how significant this can be for models with fixed charge constraints, we ran an eight product, six time period (a modification of test problem *tvw4* described in Eppen and Martin [139]) dynamic lot sizing problem in LINDO with and without the TITAN command. With the TITAN command, 476 branches and 2867 pivots are required to find the optimal integer solution and prove it is optimal. This took approximately 5 seconds. Without the TITAN command, we terminated the run after 200,000 pivots without proving optimality. The computing was done on a 200 Mhz Pentium Pro based machine running Windows NT Workstation 4.0.

Finally, observe that for time period 1, the demand constraints  $x_{i1} - I_{i1} = d_{i1}$  imply the canonical constraints (15.7)  $x_{i1} - I_{i1} \geq d_{i1}$ . Using (15.16) gives lower bounds of  $d_{i1}$  on  $x_{i1}$ . Using the fixed charge constraint  $My_{i1} - x_{i1} \geq 0$  with the new lower bounds on  $x_{i1}$  and (15.7) gives a lower bound of 1 on  $y_{i1}$  because a nonzero lower bound on a binary variable implies that the variable must be fixed to one. Tightening the lower and upper bounds and improving big  $M$  has a huge effect on the computation time of these models.

Generating lower and upper bounds may result in implied upper and lower bounds on a continuous variable  $x_j$  that are at least as good as the bounds input by the user. When this is the case, and  $x_j$  appears in an equality constraint it can be considered a *free* or *definitional variable* and substituted out of the model.

The lower and upper bound generation method described in this section considers the constraints one at a time. Obviously, by considering more than one constraint simultaneously, tighter bounds can be generated. However, one must trade off the benefit of tighter bounds versus the extra effort of considering more than one constraint simultaneously.

#### 15.2.4 Feasibility and Redundancy

After tightening the upper and lower bounds it is possible to make simple feasibility and redundancy tests. First, consider the feasibility problem. Given the canonical form (15.7), this constraint is infeasible if

$$\sum_{j \in I^+} a_j h_j + \sum_{j \in C^+} a_j h_j - \sum_{j \in I^-} a_j l_j - \sum_{j \in C^-} a_j l_j < b.$$

Of course, even if every constraint is feasible when considered alone, the linear programming relaxation of (MIP) may be infeasible. In this case, it is desirable to find an *irreducible inconsistent system* IIS for the linear programming relaxation. An IIS is a subset of infeasible linear inequalities such that if one of the inequalities is deleted the remaining system is feasible. This problem has been studied by numerous authors. See for example, Chinneck and Dravnieks [89], Gleeson and Ryan [185], Greenberg [209], and van Loon [292]. Given the system  $Ax \geq b$ , every IIS of the system can be identified using projection. See Exercise 2.22 in Chapter 2 for a characterization of IIS in terms of extreme rays of the cone  $C = \{u \mid A^\top u = 0, u \geq 0\}$ .

Next, consider redundancy. Given the canonical form (15.7), this constraint is redundant if

$$\sum_{j \in I^+} a_j l_j + \sum_{j \in C^+} a_j l_j - \sum_{j \in I^-} a_j h_j - \sum_{j \in C^-} a_j h_j \geq b.$$

Again, it is of interest to look at more than one constraint simultaneously. Perhaps the easiest extension to multiple constraints is a check for equalities (inequalities) that are a non-zero (positive) multiple of other equalities (inequalities). See the paper by Tomlin and Welch [433] and subsequent paper by Bixby and Wagner [59]. Bixby and Wagner present an  $O(\sum_j m_j \log_2(m_j))$  algorithm for detecting duplicate rows where  $m_j$  is the number of nonzeros in column  $j$  of the constraint matrix.

If the linear programming constraint set is  $Ax \geq b, x \geq 0$ , then a row (15.7) is redundant if and only if

$$\begin{aligned} \min \quad & \sum_{j \in I^+} a_j x_j + \sum_{j \in C^+} a_j x_j - \sum_{j \in I^-} a_j x_j - \sum_{j \in C^-} a_j x_j \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

has an optimal solution value greater than or equal to  $b$ . (In solving this linear program delete from the constraint set  $Ax \geq b$  the row being tested in the objective function.) It is not practical to solve a linear program to test if each row is redundant. For more effective simplex based algorithms see the survey paper by Telgen [422].

### 15.2.5 Projection and Inverse Projection

The number of variables, constraints, and the density of the constraint matrix in the system  $Ax = b$ ,  $x \geq 0$  affect the efficiency of a simplex or barrier algorithm. In Chapter 2 (3) we developed projection (inverse projection) algorithms for replacing variables (constraints) with constraints (variables). Applying projection or inverse projection to the system  $Ax = b$ ,  $x \geq 0$  prior to a barrier or simplex solution might be desirable if we can reduce the size of the system (either variables or constraints) without adversely affecting problem density.

Let  $m_i$  denote the number of nonzero elements in row  $i$  of matrix  $A$ ,  $m_i^+$  the number of positive nonzero elements in row  $i$  of matrix  $A$ , and  $m_i^-$  the number of negative nonzero elements in row  $i$  of matrix  $A$ . We discuss the effect of inverse projection (replacing constraints with variables) and leave projection (replacing variables with constraints) to the reader. In order to apply inverse projection assume the constraint set has the canonical form used in Chapter 3, namely  $Ax - bx_0 = 0$ ,  $x_0 = 1$ ,  $x \geq 0$ . Consider applying inverse projection on row  $i$ . The number of new variables required to project out row  $i$  is

$$(m_i^+ + 1)m_i^- \text{ if } b_i < 0, \quad m_i^+(m_i^- + 1) \text{ if } b_i > 0, \quad m_i^+m_i^- \text{ if } b_i = 0.$$

The effect of inverse projection of row  $i$  on the matrix density depends on the pattern on nonzero coefficients in rows other than  $i$  of the variables that had nonzero coefficients in row  $i$ . In the simplest case there are two nonzero elements in row  $i$  not including  $b_i$ . In this case several conclusions are possible depending on the sign of the nonzero elements.

**Case 1.**  $b_i = 0$  : If  $m_i^+ = 2, m_i^- = 0$ , or  $m_i^+ = 0, m_i^- = 2$  delete from the model the two variables with nonzero coefficients in row  $i$ . If  $m_i^+ = m_i^- = 1$  project out row  $i$  by making the substitution  $x_{j_1} = (a_{i,j_2}/a_{i,j_1})x_{j_2}$  for every instance of  $x_{j_1}$  in the model. This results in a model with one fewer variable and constraint and the number of nonzeros is reduced by two. Here we assume that the two nonzero coefficients in row  $i$  are  $a_{i,j_1}$  and  $a_{i,j_2}$ .

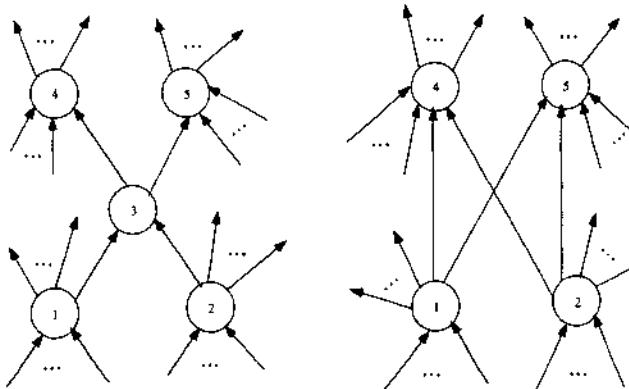
**Case 2.**  $b_i > 0$  : If  $m_i^+ = 0, m_i^- = 2$  the model is infeasible. If  $m_i^+ = m_i^- = 1$  or  $m_i^+ = 2, m_i^- = 0$  project out row  $i$  by making the substitution  $x_{j_1} = b_i/a_{i,j_1} - (a_{i,j_2}/a_{i,j_1})x_{j_2}$  for every instance of  $x_{j_1}$  in the model. This results in a model with one fewer variable and constraint and the number of nonzeros is reduced by two. Here we assume that the two nonzero coefficients in row  $i$  are  $a_{i,j_1}$  and  $a_{i,j_2}$ .

**Case 3:**  $b_i < 0$  Multiply the constraint by -1 and apply the analysis for Case 2.

Also easy is the case where  $m_i^+ = 1$ ,  $m_i^- \geq 2$  and  $b_i \geq 0$ . In this case applying the inverse projection method of Chapter 3 results in a model with one fewer rows, and one fewer variables. However, depending on the nonzero pattern we may now increase the number of nonzeros in the model. A similar analysis applies to the  $m_i^+ \geq 2$ ,  $m_i^- = 1$  and  $b_i \leq 0$ . When  $m_i^+ \geq 2$  and  $m_i^- \geq 2$  more careful analysis is required. Consider the following example.

**Example 15.5 (Network Flows)** *Applying inverse projection to a node-arc network model has the effect of replacing nodes and arcs with paths. Consider the effect of projecting out the constraint associated with node 3 in Figure 15.2. The result is to delete one constraint and four variables and create four new variables. The number of nonzeros does not change. In this example it is advantageous to make the substitution.*

Figure 15.2 Inverse Projection of Network Model



An idea closely related to inverse projection is *variable splitting*. In Chapter 13 we saw that if  $A$  had even a single dense column then  $AA^\top$  could be very dense making Cholesky factorization very difficult. Adler et al. [2] first discussed improving the sparsity of Cholesky factorization by improving the sparsity of  $A$ . See also Chang and McCormick [87] and McCormick [320] for more general results on improving the sparsity of  $A$ . More recently Gondzio [195, 194] and Vanderbei [442] propose the idea of variable splitting.

Assume column  $a^j$  is very dense. Split  $a^j$  into  $h$  columns,  $\alpha^{j_1}, \dots, \alpha^{j_h}$ , such that  $a^j = \sum_{k=1}^h \alpha^{j_k}$ . In the model formulation replace  $a^j x_j$  with  $\sum_{k=1}^h \alpha^{j_k} x'_{j_k}$  and add the constraints

$$x'_{j_k} - x'_{j_{k+1}} = 0, \quad k = 1, \dots, h-1. \quad (15.18)$$

Applying inverse projection to the constraints (15.18) results in the original model. The number of nonzeros has not increased but the pattern in the revised model is more advantageous for Cholesky factorization.

In this subsection, we assume that the variable substitutions used to project out constraints were with continuous variables. See Chapter 9 for inverse projection of constraints involving integer variables..

### 15.2.6 Probing and Variable Fixing

In a mixed 0/1 linear program *probing* refers to fixing a binary variable  $x_k$  to 0 or 1 and then observing any resulting implications. Assume variable  $x_k$  is binary. If  $x_k = 1$ ,  $k \in I^-$  and

$$\sum_{j \in I^+} a_j h_j + \sum_{j \in C^+} a_j h_j - \sum_{j \in I^- \setminus \{k\}} a_j l_j - \sum_{j \in C^-} a_j l_j < b + a_k \quad (15.19)$$

then the model is infeasible which implies it is valid to fix variable  $x_k$  to 0. If  $x_k = 0$ ,  $k \in I^+$  and

$$\sum_{j \in I^+ \setminus \{k\}} a_j h_j + \sum_{j \in C^+} a_j h_j - \sum_{j \in I^-} a_j l_j - \sum_{j \in C^-} a_j l_j < b \quad (15.20)$$

then the model is infeasible which implies it is valid to fix variable  $x_k$  to 1.

More sophisticated probing methods are available for deriving logical implications. For example, if both  $k_1, k_2 \in I^-$  are binary and

$$\sum_{j \in I^+} a_j h_j + \sum_{j \in C^+} a_j h_j - \sum_{j \in I^- \setminus \{k_1, k_2\}} a_j l_j - \sum_{j \in C^-} a_j l_j < b + a_{k_1} + a_{k_2} \quad (15.21)$$

then  $x_{k_1} = 1 \rightarrow x_{k_2} = 0$  and  $x_{k_2} = 1 \rightarrow x_{k_1} = 0$ . This implies the constraint  $x_{k_1} + x_{k_2} \leq 1$  is valid. Logical implications of this form are used to derive *clique inequalities*. See Johnson and Padberg [253].

### 15.2.7 Other Preprocessing Techniques

There are many other important preprocessing techniques. In doing the coefficient reduction we have considered the constraints one at a time. However, it is possible to improve the process by looking at multiple constraints with special structure. See Dietrich and Escudero [122] for coefficient reduction with variable upper bound side constraints and Hoffman and Padberg [235] for coefficient reduction with special ordered set side constraints.

Poorly scaled problems may cause poor performance from either a simplex or barrier algorithm. For work on scaling prior to a linear programming solution see Subsection 6.7.3 in Chapter 6 and Tomlin [432].

One can apply the methods of this section (for the linear programming relaxation) to the dual problem. Dual information such as which dual constraints are redundant is used to derive primal implications. See Luenberger [296] for detecting null variables (variable  $x_i$  is null if it is equal to zero in every solution) nonextremal variables (a variable  $x_i$  is nonextremal if the corresponding nonnegativity constraint  $x_i \geq 0$  is redundant).

## 15.3 CUTTING PLANES

By the mixed integer finite basis theorem, if  $\Gamma = \{x \in \Re^n \mid Bx \geq d, x \geq 0, x_j \in \mathcal{Z}, j \in I\}$  then  $\text{conv}(\Gamma)$  is a polyhedron. This implies there exists a system of inequalities  $(\alpha^i)^\top x \geq \alpha_{i0}$ ,  $i = 1, \dots, q$  such that  $\text{conv}(\Gamma) = \{x \mid \alpha^i x \geq \alpha_{i0}, i = 1, \dots, q\}$ . If one could even partially characterize  $\text{conv}(\Gamma)$  by finding a subset of the inequalities in the system  $\alpha^i x \geq \alpha_{i0}$ ,  $i = 1, \dots, q$ , it might help in reducing the amount of enumeration in a linear programming based branch-and-bound algorithm. An inequality  $(\alpha^i)^\top x \geq \alpha_{i0}$  is a *valid inequality* of  $\Gamma$  if  $(\alpha^i)^\top x \geq \alpha_{i0}$  for all  $x \in \Gamma$ . Valid inequalities are also called *cutting planes* or just *cuts*. The idea is to find valid inequalities so that

$$\text{conv}(\Gamma) \subseteq \{x \in \Re^n \mid Bx \geq d, x \geq 0, (\alpha^i)^\top x \geq \alpha_{i0}, i = 1, \dots, s \leq q\} \subset \bar{\Gamma}.$$

In the next subsection we look at methods for generating valid cuts for pure integer linear programs. In the second subsection we look at methods for generating valid cuts for mixed 0/1 linear programs.

### 15.3.1 Pure Integer Case

The ideas of rounding and coefficient reduction were introduced in the previous section on preprocessing. In this subsection, these ideas are used with *constraint aggregation* in order to generate valid cuts. The concept of constraint aggregation was first introduced in Chapter 2 on projecting systems of linear inequalities. Consider the system of equalities

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

Assign a set of multipliers  $u_i$ ,  $i = 1, \dots, m$  to the constraints and generate the aggregate constraint

$$\sum_{j=1}^n (\sum_{i=1}^m u_i a_{ij})x_j = \sum_{i=1}^m u_i b_i.$$

A key idea is that any solution which satisfies  $\sum_{j=1}^n a_{ij}x_j = b_i$  for  $1, \dots, m$  will satisfy the aggregate constraint  $\sum_{j=1}^n (\sum_{i=1}^m u_i a_{ij})x_j = \sum_{i=1}^m u_i b_i$ . If all of the variables  $x_j$  are required to be nonnegative, then applying rounding to the aggregate constraint gives the valid cut

$$\sum_{j=1}^n \lfloor (\sum_{i=1}^m u_i a_{ij}) \rfloor x_j \leq \sum_{i=1}^m u_i b_i.$$

If each  $x_j$  is a nonnegative and integer, then it is also valid to round down the right hand side which gives the cut

$$\sum_{j=1}^n \lfloor (\sum_{i=1}^m a_{ij}) \rfloor x_j \leq \lfloor \sum_{i=1}^m u_i b_i \rfloor. \quad (15.22)$$

The cut (15.22) is a Chvátal-Gomory (C-G) cut and the procedure used to derive the cut is Chvátal-Gomory rounding. See Chvátal [92] and Gomory [191, 193]. When all of the variables are integer, C-G rounding is used to generate a cut from the linear programming solution to  $(\overline{MIP})$ . Following the notation in Chapter 5 and assuming  $m$  basic variables, the updated tableau is

$$x_{B_i} + \sum_{j \in N} \bar{a}_{ij}x_j = \bar{b}_i, \quad i = 1, \dots, m \quad (15.23)$$

where  $B_i$ ,  $i = 1, \dots, m$ , indexes the basic variables and  $N$  indexes the non basic variables. Applying C-G rounding gives

$$x_{B_i} + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor, \quad i = 1, \dots, m. \quad (15.24)$$

Multiplying (15.24) by -1 and adding to (15.23) gives

$$\sum_{j \in N} \bar{f}_{ij} x_j \geq \bar{f}_{i0} \quad i = 1, \dots, m \quad (15.25)$$

where  $\bar{f}_{ij} = \bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor$  and  $\bar{f}_{i0} = \bar{b}_i - \lfloor \bar{b}_i \rfloor$ . It is easy to see that the basic feasible solution which satisfies (15.23) does not satisfy (15.25). The cut given in (15.25) is a *Gomory cut* and one of the earliest cuts used in integer programming. The earliest algorithms for solving integer programs were cutting plane algorithms developed in the 1950s. Unfortunately, there are several problems with Gomory cuts.

1. Gomory cuts are dense (there is a potential nonzero coefficient for each nonbasic variable) and significantly increase the number of nonzeros in the problem.
2. Gomory cuts often cause roundoff problems.
3. A Gomory cut may not be “deep,” or it may not be a face of  $\text{conv}(\Gamma)$ .

Despite these problems, Gomory did provide a proof of a finite cutting plane algorithm based on (15.25). However, because the Gomory cuts were not effective in practice, new methods were sought for solving integer programs. This led to the development of linear programming based branch-and-bound algorithms in the 1960s. For problems with large integrality gaps, linear programming based branch-and-bound is not effective. The search for bounding methods superior to the linear programming relaxation led to the development of Lagrangian dual methods in the 1970s. Then, in the early 1980s, spurred by the work of Crowder, Johnson, and Padberg [103] there was a return to cutting plane methods in conjunction with some of the preprocessing techniques developed in the previous section. Unlike the Gomory cuts, the cuts used by Crowder, Johnson, and Padberg were based on polyhedral theory. Assume the constraints defining the integer program are  $(a^i)^\top x \geq b_i$ ,  $i = 1, \dots, m$ , plus nonnegativity and integrality of  $x$ . Define

$$\Gamma_i := \{x \in \mathbb{R}^n \mid (a^i)^\top x \geq b_i, x \geq 0, x \in \mathbb{Z}^n\}.$$

Then  $\text{conv}(\cap_{i=1}^m \Gamma_i) \subseteq \cap_{i=1}^m \text{conv}(\Gamma_i)$ . If the constraints are extremely sparse so that very few non zero coefficients in  $a^i$  appear in  $a^j$  then  $\cap_{i=1}^m \text{conv}(\Gamma_i)$  will be a good approximation of  $\text{conv}(\cap_{i=1}^m \Gamma_i)$ . If the  $x$  vector is required to be binary then each  $\Gamma_i$  is the solution set of a binary knapsack problem. Using results of Balas [21] characterizing certain faces of knapsack polyhedra, Crowder, Johnson, and Padberg developed a very effective class of cuts for pure

binary problems. We first derive a broad class of valid cuts for pure integer programming problems by combining constraint aggregation with the coefficient reduction process developed in the previous section. We then show how an important set of constraints used by Crowder, Johnson, and Padberg are a special case of our broad class. Consider the constraints

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad i = 1, \dots, m$$

and again assign a set of multipliers  $u_i$ ,  $i = 1, \dots, m$  to these constraints and generate the aggregate constraint

$$\sum_{j=1}^n (\sum_{i=1}^m u_i a_{ij})x_j = \sum_{i=1}^m u_i b_i.$$

If all of the variables  $x_j$  are required to be nonnegative, then a valid cut based on the aggregate constraint is

$$\sum_{j=1}^n \max \{0, (\sum_{i=1}^m u_i a_{ij})\}x_j \geq \sum_{i=1}^m u_i b_i.$$

If every  $x_j$  is a nonnegative integer variable and if  $\sum_{i=1}^m u_i b_i > 0$ , then a valid *constraint aggregation, coefficient reduction* (CACR) cut is

$$\sum_{j=1}^n \min (\sum_{i=1}^m u_i b_i, \max \{0, (\sum_{i=1}^m u_i a_{ij})\})x_j \geq \sum_{i=1}^m u_i b_i. \quad (15.26)$$

This can be specialized to the constraints which define the special structure  $\Gamma_i$ . Any inequality in a pure 0/1 linear program can be written as

$$\sum_{j \in I^+} a_j x_j - \sum_{j \in I^-} a_j x_j \geq b \quad (15.27)$$

where  $a_j > 0$  for  $j \in I^+ \cup I^-$  and  $x_j \in \{0, 1\}$  for all  $j \in I^+ \cup I^-$ . Since  $x_j$  is a binary variable the simple upper bound constraints  $x_j \leq 1$  for all  $j \in W \subset I^+$  are valid *side constraints*. Assign a multiplier of  $-a_j$  to all of the side constraints  $x_j \leq 1$  for  $j \in W$  and a multiplier of  $+1$  to the constraint (15.27). The aggregate constraint is

$$\sum_{j \in I^+ \setminus W} a_j x_j - \sum_{j \in I^-} a_j x_j \geq \lambda := b - \sum_{j \in W} a_j.$$

If  $\lambda = b - \sum_{j \in W} a_j > 0$ , then the valid cut corresponding to (15.26) is

$$\sum_{j \in I^+ \setminus W} \min\{\lambda, a_j\} x_j \geq \lambda. \quad (15.28)$$

The primary special structure cut for  $\Gamma$ , used by Crowder, Johnson, and Padberg was the minimal cover cut. The set  $I^+ \setminus W$  is a *minimal cover* (see Balas [21] and Balas and Jeroslow [28] for earlier work on canonical inequalities) with respect to  $\sum_{j \in I^+} a_j x_j \geq b$  if  $b - \sum_{j \in W} a_j > 0$  and  $a_j \geq b - \sum_{j \in W} a_j > 0$  for all  $j \in I^+ \setminus W$ .

The number of possible minimal cover cuts for a single knapsack constraint with  $n$  variables is an exponential function of  $n$ . Therefore, it is not desirable to generate all the possible minimal cover cuts a priori. As with Benders' decomposition, we generate the cuts as needed, or on-the-fly. This requires a method for answering the *separation problem*: "given  $\bar{x} \in \mathbb{R}^n$ , is  $\bar{x} \in \text{conv}(\Gamma)$ , and if not, find a cut  $\alpha^\top x \geq \alpha_0$  such that  $\alpha^\top \bar{x} < \alpha_0$  and  $\alpha^\top x \geq \alpha_0$  for all  $x \in \text{conv}(\Gamma)$ ." For more on the separation problem and its importance in combinatorial theory see Grötschel, Lovász, and Schrijver[212]. Crowder, Johnson, and Padberg show that for  $\Gamma$ , the separation problem is actually a knapsack problem. If the cut is (15.28), given  $\bar{x} > 0$ , finding the "most violated cut" implies finding a  $W \subset I^+$  so that  $\sum_{j \in I^+ \setminus W} \min\{\lambda, a_j\} \bar{x}_j - \lambda$  is minimized and  $\lambda = b - \sum_{j \in W} a_j$  is strictly positive. This is done by solving a binary knapsack problem. Let  $r_j = 1$  if  $j \in I^+ \setminus W$  and 0 otherwise. Instead of requiring  $\lambda = b - \sum_{j \in W} a_j$ , (15.28) is valid as long as  $\lambda \leq b - \sum_{j \in W} a_j$ . Then the separation knapsack problem is

$$(KSEP) \quad \begin{aligned} \min \quad & \sum_{j \in I^+} \min\{\lambda, a_j\} \bar{x}_j - \lambda \\ \text{s.t.} \quad & b - \sum_{j \in I^+} a_j + \sum_{j \in I^+} a_j r_j \geq \lambda \\ & r_j \in \{0, 1\}, \quad j \in I^+. \end{aligned}$$

Clearly, for any  $\lambda > 0$ , when the optimal solution value of (KSEP) is negative, the current fractional solution  $\bar{x}$  does not satisfy the cut (15.28). When  $\lambda = 1$ , the cut is a *minimal cover cut* and the following simple lemma is left as Exercise 15.3.

**Lemma 15.6** *Given integer  $a^i, b$ , there exists a minimal cover cut which cuts off  $\bar{x}$  if and only if the optimal solution value of (KSEP) is negative for  $\lambda = 1$ .*

**Example 15.7** In this example we generate minimal cover cuts for the generalized assignment problem. The problem was first introduced in Subsection 1.3.3 in Chapter 1.

```

MIN      2 X11 + 11 X12 + 7 X21 + 7 X22 + 20 X31 +
2 X32 + 5 X41 + 5 X42
SUBJECT TO
 2) X11 + X12 =    1
 3) X21 + X22 =    1
 4) X31 + X32 =    1
 5) X41 + X42 =    1
 6) 3 X11 + 6 X21 + 5 X31 + 7 X41 <=   13
 7) 2 X12 + 4 X22 + 10 X32 + 4 X42 <=   10
END
OBJECTIVE FUNCTION VALUE
 1) 20.32000
VARIABLE      VALUE      REDUCED COST
X11      1.000000      .000000
X21      1.000000      .000000
X31      .240000       .000000
X32      .760000       .000000
X41      .400000       .000000
X42      .600000       .000000

```

Solve a separation problem for each of the knapsack constraints in 6) and 7) which correspond to the machine capacities. First convert these constraints into the canonical  $\geq$  form (15.27) by complementing the variables  $z_{ij} = 1 - x_{ij}$  and then multiplying by -1. In canonical form 6) and 7) are

$$\begin{aligned} 3z_{11} + 6z_{21} + 5z_{31} + 7z_{41} &\geq 8 \\ 2z_{12} + 4z_{22} + 10z_{32} + 4z_{42} &\geq 10 \end{aligned}$$

The separation problem for 6) with  $\lambda = 1$  is

$$(KSEP) \quad \begin{aligned} \min \quad & 0r_{11} + 0r_{21} + .76r_{31} + .6r_{41} - 1 \\ \text{s.t.} \quad & 3r_{11} + 6r_{21} + 5r_{31} + 7r_{41} \geq 14 \\ & r_{11}, r_{21}, r_{31}, r_{41} \in \{0, 1\} \end{aligned}$$

The optimal solution is  $r_{11} = 1$ ,  $r_{21} = 1$ ,  $r_{31} = 0$  and  $r_{41} = 1$ . The implied minimal cover cut is  $z_{11} + z_{21} + z_{41} \geq 1$  which is  $x_{11} + x_{21} + x_{41} \leq 2$  in the original variables. The current linear programming relaxation violates this cut. Similarly the implied minimal cover cut for the second machine is  $x_{32} + x_{42} \leq 1$ . Add these cuts to the linear programming relaxation and resolve.

#### OBJECTIVE FUNCTION VALUE

1) 23.20000

VARIABLE	VALUE	REDUCED COST
X11	1.000000	.000000
X22	1.000000	.000000
X31	.400000	.000000
X32	.600000	.000000
X41	1.000000	.000000

The objective function value of the linear programming relaxation increases from 20.32 to 23.2 but the solution is still fractional. Applying separation to generate a knapsack constraint yields for the first machine gives

$$\begin{aligned} \min \quad & 0r_{11} + 1r_{21} + .6r_{31} + 0r_{41} \\ (\text{KSEP}) \quad \text{s.t.} \quad & 3r_{11} + 6r_{21} + 5r_{31} + 7r_{41} \geq 14 \\ & r_{11}, r_{21}, r_{31}, r_{41} \in \{0, 1\} \end{aligned}$$

The optimal solution is  $r_{11} = 1$ ,  $r_{21} = 0$ ,  $r_{31} = 1$  and  $r_{41} = 1$ . The implied minimal cover cut is  $z_{11} + z_{31} + z_{41} \geq 1$ , which is  $x_{11} + x_{31} + x_{41} \leq 2$  in the original variables. The current linear programming relaxation violates this cut. Similarly the implied minimal cover cut for the second machine is  $x_{22} + x_{32} \leq 1$ . Add these cuts to the linear programming relaxation and resolve.

1) 25.00000

VARIABLE	VALUE	REDUCED COST
X11	0.666667	0.000000
X12	0.333333	0.000000
X21	0.666667	0.000000
X22	0.333333	0.000000
X31	0.333333	0.000000

X32	0.666667	0.000000
X41	0.666667	0.000000
X42	0.333333	0.000000

The optimal solution value increases to 25. The solution is still fractional, but there are no violated minimal covers. Why?

Numerous other cuts exist for pure 0/1 linear programs. Crowder, Johnson, and Padberg also added  $(1-k)$  configuration cuts in their landmark study. Clique cuts have also proved useful. See Johnson and Padberg [253]. Later we show how to strengthen cuts through a lifting procedure. We have shown how to generate cuts from G-G rounding and from CACR. An interesting question is: “given a valid cut  $\alpha^\top x \geq \alpha_0$ , for  $\Gamma = \{x \in \mathbb{Z}^n \mid Ax \geq b, x \geq 0\}$ , can this cut be generated by C-G rounding or CACR.” The answer is: “given a valid cut  $\alpha^\top x \geq \alpha_0$ , for  $\Gamma = \{x \in \mathbb{Z}^n \mid Ax \geq b, x \geq 0\}$ , then repeated application of CACR can generate the cut  $\alpha^\top x \geq \beta_0$  where  $\beta_0 \geq \alpha_0$ . This result is in Martin and Schrage [319]. A similar result is available for C-G rounding. See Chvátal [92].

Although C-G rounding and CACR are valid for general integer programs, most applications involve 0/1 integer variables. Indeed, most of the computational studies in the literature with integer variables are with 0/1 variables. Other than Ceria et al. [85] and Cook et al. [100], we are not aware of computational studies where the majority of the problems solved are large scale linear programs with general integer variables.

### 15.3.2 Mixed Integer Cuts

In this section we extend the ideas for pure integer programming to the mixed case. Obviously, the concept of constraint aggregation is still valid in the mixed case. However, if continuous variables are present, then it is *not valid* to conclude that

$$\sum_{j=1}^n \lfloor \sum_{i=1}^m u_i a_{ij} \rfloor x_j \leq \sum_{i=1}^m u_i b_i \quad \rightarrow \quad \sum_{j=1}^n \lfloor \sum_{i=1}^m u_i a_{ij} \rfloor x_j \leq \lfloor \sum_{i=1}^m u_i b_i \rfloor.$$

Nemhauser and Wolsey [350, 355] have shown how to extend C-G cuts to the mixed integer case with their mixed integer rounding (MIR) cuts. Rather than follow Nemhauser and Wolsey, we develop the mixed integer cuts using constraint aggregation and coefficient reduction since this extends directly without

any changes to the mixed case. If the integer variables are indexed by  $I$  and the continuous variables by  $C$ , then after aggregating constraints, it follows that a valid CACR cut is

$$\begin{aligned} \sum_{j \in I} \min \left( \sum_{i=1}^m u_i b_i, \max \left\{ 0, \left( \sum_{i=1}^m u_i a_{ij} \right) \right\} \right) x_j + \\ \sum_{j \in C} \max \left\{ 0, \left( \sum_{i=1}^m u_i a_{ij} \right) \right\} x_j \geq \sum_{i=1}^m u_i b_i. \quad (15.29) \end{aligned}$$

In constraint (15.29) we simply apply coefficient reduction to the integer variables and do not alter the coefficients of the continuous variables. We are assuming that the integer variables are required to be nonnegative and that  $\sum_{i=1}^m u_i b_i > 0$ . Following the development in the pure integer case, we develop CACR cuts for a very special mixed 0/1 structure. In particular, we generate cuts based on our constraint in canonical form

$$\sum_{j \in I^+} a_j x_j + \sum_{j \in C^+} a_j x_j - \sum_{j \in I^-} a_j x_j - \sum_{j \in C^-} x_j \geq b$$

where the  $x_j$  are continuous for all  $j \in C^+ \cup C^-$ , integer for all  $j \in I^+ \cup I^-$  and all  $a_j$  are positive. This general canonical constraint is the *parent constraint*. Many applications which are modeled as a mixed integer 0/1 linear program have subsets of constraints with the special structure where there is a parent constraint and several associated *side or fixed charge constraints*. In the pure 0/1 case, the side constraints were simple upper bound constraints of the form  $x_j \leq 1$ . In the mixed case we generalize the structure of the side constraints to

$$\sum_{j \in G_k} a_{kj} x_j - \sum_{j \in E_k} a_{kj} y_j \leq b_k$$

which is a *positive side constraint* if

1.  $a_{kj} > 0$  for all  $j \in (G_k \cup E_k)$ ,
2.  $b_k \geq 0$ ,
3.  $y_j \in \{0, 1\}$  for all  $j \in E_k$ ,
4.  $C^+ \cap G_k \neq \emptyset$ ,  $I^+ \cap G_k = \emptyset$
5.  $(I^- \cup C^-) \cap G_k = \emptyset$ ,
6.  $(I^+ \cup I^- \cup C^+ \cup C^-) \cap E_k = \emptyset$ ,

and a *negative side constraint* if conditions 4 and 5 are replaced by  $C^- \cap G_k \neq \emptyset$ ,  $I^- \cap G_k = \emptyset$  and  $(I^+ \cup C^+) \cap G_k = \emptyset$ , respectively. For a given parent constraint, assume a set of positive side constraints indexed by  $M^+$  and a set of negative side constraints indexed by  $M^-$  have been identified. For the positive and negative side constraints define the following multipliers

$$\begin{aligned}\beta_k^+ &:= \max \{a_j/a_{kj} \mid j \in C^+ \cap G_k\}, \\ \beta_k^- &:= \max \{a_j/a_{kj} \mid j \in C^- \cap G_k\}.\end{aligned}$$

In order to generate a CACR cut assign a multiplier of 1 to the parent constraint, multipliers of  $-\beta_k^+$  to the positive side constraints, multipliers of  $\beta_k^-$  to the negative side constraints, multipliers of  $-\beta_k^+ a_{kj}$  to each simple upper bound constraint  $y_j \leq 1$  for all  $j \in V \subseteq (\cup_{M^+} E_k)$  and  $-a_j$  to each simple upper bound constraint  $x_j \leq 1$  for all  $j \in W \subseteq I^+$ . In summary, the constraints and corresponding multipliers are

$$\begin{aligned}\sum_{j \in I^+} a_j x_j + \sum_{j \in C^+} a_j x_j - \sum_{j \in I^-} a_j x_j - \sum_{j \in C^-} x_j &\geq b, \quad 1 \\ \sum_{j \in G_k} a_{kj} x_j - \sum_{j \in E_k} a_{kj} y_j &\leq b_k, \quad -\beta_k^+, \quad k \in M^+ \\ \sum_{j \in G_k} a_{kj} x_j + \sum_{j \in E_k} a_{kj} (1 - y_j) + s_k &= b_k + \sum_{j \in E_k} a_{kj}, \quad \beta_k^-, \quad k \in M^- \\ y_j &\leq 1, \quad -\beta_k^+ a_{kj}, \quad j \in V \subseteq (\cup_{M^+} E_k) \\ x_j &\leq 1, \quad -a_j, \quad j \in W \subseteq I^+.\end{aligned}$$

In the negative side constraints we have complemented the 0/1 variables and converted to equality constraints by adding the slack variable  $s_k$ . Using these multipliers aggregate into the constraint

$$\begin{aligned}&\sum_{I^+ \setminus W} a_j x_j + \sum_{j \in S} a_j x_j + \sum_{k \in M^+} \sum_{j \in G_k \cap C^+} (a_j - \beta_k^+ a_{kj}) x_j \\ &\quad - \sum_{k \in M^+} \beta_k^+ \sum_{j \in G_k \setminus C^+} a_{kj} x_j + \sum_{k \in M^+} \beta_k^+ \sum_{j \in E_k \setminus V} a_{kj} y_j \\ &+ \sum_{k \in M^-} \sum_{j \in G_k \setminus C^-} \beta_k^- a_{kj} x_j - \sum_{j \in Q} a_j x_j + \sum_{k \in M^-} \sum_{j \in G_k \cap C^-} (\beta_k^- a_{kj} - a_j) x_j \\ &\quad + \sum_{k \in M^-} \beta_k^- \sum_{j \in E_k} a_{kj} (1 - y_j) + \sum_{k \in M^-} \beta_k^- s_k \geq \lambda\end{aligned}$$

where  $S = (C^+ \setminus \cup_{k \in M^+} G_k)$ ,  $Q = (C^- \setminus \cup_{k \in M^-} G_k)$  and

$$\lambda := b - \sum_{j \in W} a_j - \sum_{j \in V} \beta_k^+ a_{kj} - \sum_{k \in M^+} \beta_k^+ b_k + \sum_{k \in M^-} \beta_k^- (b_k + \sum_{j \in E_k} a_{kj}) \quad (15.30)$$

Delete the terms with negative coefficients and perform coefficient reduction on the integer variables.

$$\begin{aligned} & \sum_{I^+ \setminus W} \min\{\lambda, a_j\} x_j + \sum_{j \in S} a_j x_j + \sum_{k \in M^+} \sum_{j \in E_k \setminus V} \min\{\lambda, \beta_k^+ a_{kj}\} y_j + \\ & \sum_{k \in M^-} \sum_{j \in G_k \setminus C^-} \beta_k^- a_{kj} x_j + \sum_{k \in M^-} \sum_{j \in G_k \cap C^-} (\beta_k^- a_{kj} - a_j) x_j \\ & + \sum_{k \in M^-} \sum_{j \in E_k} \min\{\lambda, \beta_k^- a_{kj}\} (1 - y_j) + \sum_{k \in M^-} \beta_k^- s_k \geq \lambda \quad (15.31) \end{aligned}$$

Substitute for  $s_k$ ,  $k \in M^-$  and get

$$\begin{aligned} & \sum_{I^+ \setminus W} \min\{\lambda, a_j\} x_j + \sum_{j \in S} a_j x_j + \sum_{k \in M^+} \sum_{j \in E_k \setminus V} \min\{\lambda, \beta_k^+ a_{kj}\} y_j \\ & - \sum_{k \in M^-} \sum_{j \in G_k \cap C^-} a_j x_j - \sum_{k \in M^-} \sum_{j \in E_k} \max\{0, (\beta_k^- a_{kj} - \lambda)\} (1 - y_j) \\ & \geq \lambda - \sum_{k \in M^-} \beta_k^- b_k - \sum_{k \in M^-} \sum_{j \in E_k} \beta_k^- a_{kj} \quad (15.32) \end{aligned}$$

$$= b - \sum_{j \in W} a_j - \sum_{j \in V} \beta_k^+ a_{kj} - \sum_{k \in M^+} \beta_k^+ b_k \quad (15.33)$$

As in the pure integer case, we are interested in generating cuts on the fly that separate a fractional solution from the convex hull of mixed integer solutions. We define an extension of the separation problem used in the pure integer case. Let  $r_j = 1$  if  $j \in I^+ \setminus W$ , 0 otherwise and  $z_k = 1$  if side constraint  $k \in (M^+ \cup M^-)$  is included in the aggregation, 0 otherwise and  $q_j = 1$  if  $j \in V \subseteq (\cup_{M^+} E_k)$ . The objective is to

$$\begin{aligned} & \min \left( \sum_{j \in I^+} \min\{\lambda, a_j\} \bar{x}_j r_j + \sum_{j \in C^+} a_j \bar{x}_j - \sum_{k \in M^+} \sum_{j \in C^+ \cap G_k} a_j \bar{x}_j z_k \right. \\ & + \sum_{k \in M^+} \sum_{j \in E_k} \min\{\lambda, \beta_k^+ a_{kj}\} \bar{y}_j z_k (1 - q_j) - \sum_{k \in M^-} \sum_{j \in G_k \cap C^-} a_j \bar{x}_j z_k \\ & \quad \left. - \sum_{k \in M^-} \sum_{j \in E_k} \max\{0, (\beta_k^- a_{kj} - \lambda)\} (1 - \bar{y}_j) z_k \right. \\ & \quad \left. - (\lambda - \sum_{k \in M^-} \beta_k^- b_k z_k - \sum_{k \in M^-} \sum_{j \in E_k} \beta_k^- a_{kj} z_k) \right) \quad (15.34) \end{aligned}$$

$$\text{s.t. } \sum_{j \in I^+} a_j r_j - \sum_{k \in M^+} \sum_{j \in E_k} \beta_k^+ a_{kj} q_j - \sum_{k \in M^+} \beta_k^+ b_k z_k + \sum_{k \in M^-} \beta_k^- b_k z_k$$

$$+ \sum_{k \in M^-} (\sum_{j \in E_k} \beta_k^- a_{kj}) z_k \geq \lambda - b + \sum_{j \in I^+} a_j. \quad (15.35)$$

As in the pure 0/1 case, as long as (15.35) holds, the CACR cut given in (15.32) is valid for any positive  $\lambda$ . In the pure 0/1 case, this problem reduces to the separation problem (*KSEP*).

**Example 15.8** This example is from Martin and Schrage [319] and is based on constraints arising in a distribution application. Consider the parent constraint

$$-x_1 - x_2 - x_3 - x_4 + 22x_5 + 44x_6 + 88x_7 \geq 0.$$

There are two negative side constraints  $x_1 + x_2 \leq 10$  and  $x_3 + x_4 \leq 20$ . In our notation,  $C^- = \{1, 2, 3, 4\}$ ,  $C^+ = I^- = \emptyset$ ,  $I^+ = \{5, 6, 7\}$ ,  $G_1 = \{1, 2\}$ ,  $G_2 = \{3, 4\}$ ,  $E_1 = \emptyset$ ,  $E_2 = \emptyset$ ,  $M^- = \{1, 2\}$ ,  $M^+ = \emptyset$ ,  $\beta_1^- = 1, \beta_2^- = 1$ ,  $b = 0, b_1 = 10$  and  $b_2 = 20$ . Assume the current linear programming relaxation solution is

$$\bar{x}_1 = 5, \bar{x}_2 = 5, \bar{x}_3 = 5, \bar{x}_4 = 15, \bar{x}_5 = 1, \bar{x}_6 = 8/44, \bar{x}_7 = 0.$$

Given a  $\lambda = 1$ , the knapsack problem for finding a violated CACR cut is

$$\begin{aligned} \min \quad & r_5 + (8/44)r_6 + (10 - 10)z_1 + (20 - 20)z_2 - 1 \\ \text{s.t.} \quad & 22r_5 + 44r_6 + 88r_7 + 10z_1 + 20z_2 \geq 155 \\ & r_5, r_6, r_7, z_1, z_2 \in \{0, 1\} \end{aligned}$$

The optimal solution is  $z_1 = z_2 = r_6 = r_7 = 1$  and  $r_5 = 0$ . Then  $\lambda = 44 + 88 + 10 + 20 - 154 = 8$  and the cut is  $x_1 + x_2 + x_3 + x_4 \leq 22 + 8x_6 + 8x_7$ . The current linear programming relaxation does not satisfy this cut.

**Example 15.9 (Generalized Flow Cover Cuts)** Van Roy and Wolsey in [439] (see also Nemhauser and Wolsey [350]) generalizing earlier work by Padberg, Van Roy, and Wolsey [364] developed generalized flow cover inequalities for the structure

$$\begin{aligned} \Gamma = \{ (x, y) | \sum_{j \in C^-} x_j - \sum_{j \in C^+} x_j \leq b, x_j \leq a_j y_j, j \in M^+ \subseteq C^+, \\ x_j \leq a_j y_j, j \in M^- \subseteq C^-, x \geq 0, y_j \leq 1, y_j \in \{0, 1\}, j \in M^+ \cup M^- \}. \end{aligned}$$

Treating  $\sum_{j \in C^-} x_j - \sum_{j \in C^+} x_j \leq b$  as the parent constraint and  $x_j \leq a_j y_j$  as the side constraints, the CACR cut (15.32) for the special structure  $\Gamma$  is

$$\begin{aligned} \sum_{C^+ \setminus M^+} x_j + \sum_{j \in M^+ \setminus V} \min\{\lambda, a_j\} y_j - \sum_{j \in M^-} x_j - \sum_{j \in M^-} \max\{0, (a_j - \lambda)\}(1 - y_j) \\ \geq -b - \sum_{j \in V} a_j. \end{aligned}$$

*Computational results with generalized flow cover cuts (and others) are given in Van Roy and Wolsey [307]. Van Roy and Wolsey [438] developed an extension of the flow cover inequalities for network flow problems where there are fixed charges on the arcs.*

**Example 15.10 (Simple Plant Location)** Consider the aggregated formulation (SPLAGG) of the simple plant location problem from Example 15.1. Treat the demand constraint  $\sum_{i=1}^n x_{ik} = 1$  as the parent constraint and the fixed charge constraint  $\sum_{j=1}^m x_{lj} \leq my_l$  as a positive side constraint. Aggregating the constraints, simplifying, and performing coefficient reduction gives the following.

$$\begin{aligned} my_l - \sum_{j=1}^m x_{lj} + \sum_{i=1}^n x_{ik} &\geq 1 \\ my_l - \sum_{\substack{j=1 \\ j \neq k}}^m x_{lj} + \sum_{\substack{i=1 \\ i \neq l}}^n x_{ik} &\geq 1 \\ y_l + \sum_{\substack{i=1 \\ i \neq l}}^n x_{ik} &\geq 1 \\ y_l \geq 1 - \sum_{\substack{i=1 \\ i \neq l}}^n x_{ik} &= x_{lk} \\ x_{lk} &\leq y_l \end{aligned}$$

This is valid for any plant index  $l$  and customer index  $k$  so CACR generates the disaggregated formulation. See Leung and Magnanti [286] for valid cuts and facets of the capacitated plant location problem.

**Example 15.11 (Dynamic Lot Sizing)** Consider the uncapacitated dynamic lot size model which is Example 15.3 simplified to only one product and no capacity constraints. Treat the demand constraint  $I_{t-1} + x_t - I_t = d_t$  as the parent constraint and the fixed charge constraint  $x_t \leq My_t$  as a positive side constraint. Assign a multiplier of 1 to the parent and -1 to the positive side constraint. Aggregating gives the constraint  $I_{t-1} + My_t - I_{t-1} \geq d_t$ . This implies  $I_{t-1} + My_t \geq d_t$  and applying coefficient reduction gives  $I_{t-1} + d_t y_t \geq d_t$ . After adding these cuts to the formulation in Example 15.3 the new linear programming relaxation value is 6467 and finding and proving integer optimality requires only 2 branches and 72 pivots.

Barany, Van Roy, and Wolsey [35] have actually characterized the convex hull of solutions to the single product, uncapacitated dynamic lot size model. By uncapacitated, we mean the production capacity in each period  $t$ , is at least  $\sum_{t=1}^T d_t$ . Adding the so called  $(l, S)$  inequalities

$$\sum_{t \in S} x_t \leq \sum_{t \in S} \left( \sum_{k=t}^l d_k \right) y_t + I_l \quad (15.36)$$

for all  $l = 1, \dots, T$  and  $S \subseteq \{1, \dots, l\}$  to the demand and fixed charge constraints results in a characterization of the convex hull of solutions to the uncapacitated dynamic lot size model. Using CACR we generated the cut,  $I_{t-1} + d_t y_t \geq d_t$ . Since  $d_t = I_{t-1} + x_t - I_t$  this CACR cut is equivalent to  $x_t \leq I_t + d_t y_t$  which is an  $(l, S)$  inequality for the case where  $S = \{l\}$ . It is left as Exercise 15.5 to show that repeated application of CACR generates all of the  $(l, S)$  cuts. Leung, Magnanti, and Vachani [287] study facets for the capacitated single product dynamic lot size model.

All of the  $(l, S)$  inequalities are generated by repeated application of CACR. Can any valid mixed cut for a mixed integer linear program be generated by CACR? That is: "given a valid cut  $\alpha^\top x \geq \alpha_0$ , for  $\Gamma = \{x \in \mathbb{R}^n \mid Ax \geq b, x \geq 0, x_j \in Z, j \in I\}$ , can this cut be generated by CACR?" The answer is: "given a valid cut  $\alpha^\top x \geq \alpha_0$ , for  $\Gamma$  and any  $\epsilon > 0$ , repeated application of CACR can generate the cut  $\alpha^\top x \geq \beta_0$  where  $\beta_0 \geq \alpha_0 - \epsilon$ ." This result is in Martin and Schrage [319]. In the case of 0/1 mixed integer programming Nemhauser and Wolsey [350, 355] show any valid inequality is an MIR inequality.

The literature on cutting planes is truly voluminous. See Jünger, Reinelt, and Thienel [251] for a good recent survey paper. The reader interested in a very thorough and deep development of cutting plane theory from an algebraic standpoint should consult Jeroslow [244, 246].

## 15.4 BRANCH-AND-CUT

Branch-and-bound and cutting planes are two methods for solving mixed integer linear programming problems. In theory, branch-and-bound will always find an optimal solution, if one exists. Cutting plane algorithms may or may not converge to an optimal solution. Obviously both of these methods can be used together. It was known for many years that adding cuts before applying enumeration could be very effective in reducing the integrality gap and the

amount of enumeration required. See, for example, the work of Crowder, Johnson, and Padberg [103]. *Branch-and-cut* is essentially a feedback loop from the branch-and-bound stage back to the preprocessing stage and is designed to take advantage of both enumeration and cutting planes. The idea is to first tighten the problem by adding cuts, and then if no more valid cuts can be found, or the cuts are having little effect improving the objective function value, begin linear programming based branch-and-bound. After integer variables have been fixed at candidate nodes, try to find more valid cuts for the candidate problems at each node. Padberg and Rinaldi in 1987 were the first to use the term branch-and-cut. However, the feedback loop appears in an earlier paper by Hoffman and Padberg [233]. In order to minimize the book keeping, and to keep the data structures as simple as possible, it is important that the cuts added at any node of the tree *are valid at every other node!* The problem of generating a cut at a node which is valid for the entire search tree is discussed further in the next section. See Figure 15.3 for a flow chart of the branch-and-cut method. Note the additional feedback loop with the branch-and-cut. Preprocessing is applied to the problem, cuts are generated and then the enumeration phase is entered. During enumeration additional cuts are added. Branch-and-cut is now part of a number of commercial codes. For computational results using branch-and-cut see Applegate, Bixby, and Cook [15], Balas, Ceria, and Cornuéjols [27], Hoffman and Padberg [234], and Padberg and Rinaldi [362, 363]. Although we do not discuss heuristics in this text, most successful implementations of branch-and-cut, use heuristic methods to find good feasible integer solutions before entering branch-and-bound. We next illustrate the branch-and-cut idea for the generalized assignment problem.

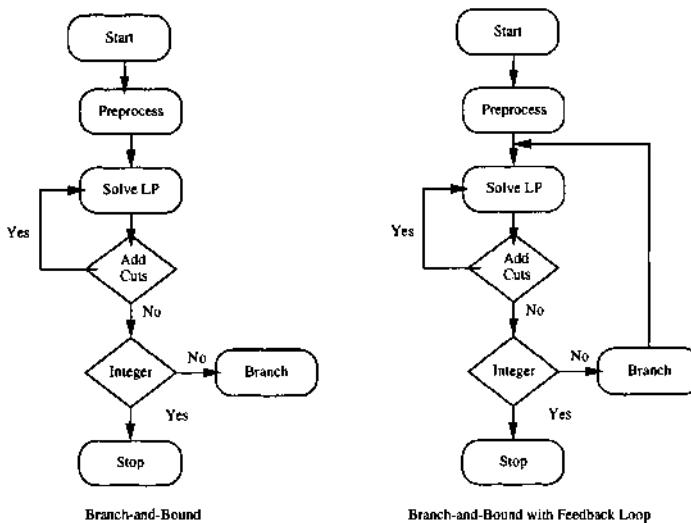
**Example 15.12 (Example 15.7 Continued)** *After four minimal cover cuts are added there is a solution value of 25 and the optimal solution is fractional. There are no more violated minimal covers for this solution. Enter the branch-and-bound stage. Select variable  $x_{11}$  for branching. Create two candidate problems, one with the added constraint  $x_{11} = 0$ , the other with  $x_{11} = 1$ . Below is the solution for the candidate problem with  $x_{11} = 0$ .*

#### OBJECTIVE FUNCTION VALUE

1) 30.04000

VARIABLE	VALUE	REDUCED COST
X12	1.000000	0.000000
X21	1.000000	0.000000
X31	0.280000	0.000000

Figure 15.3 Branch-and-Cut Flowchart



X32	0.720000	0.000000
X41	0.800000	0.000000
X42	0.200000	0.000000

Based on this solution two minimal cover cuts which are not satisfied by the current solution are

$$x_{21} + x_{31} + x_{41} \leq 2, \quad x_{12} + x_{32} \leq 1. \quad (15.37)$$

These two cuts are found by solving the separation problem (KSEP). After adding these two cuts and resolving, the optimal linear programming relaxation value for the candidate problem with  $x_{11} = 0$  is 43 and the solution is integer. No further branching from this candidate problem is required. Most importantly, the minimal cover cuts (15.37) are also valid for the candidate problem with  $x_{11} = 1$ .

## 15.5 LIFTING

During branch-and-cut valid cuts are generated at nodes of the enumeration tree. We have already stressed the importance of generating cuts which are

valid for the entire tree. Thus, if variable  $x_t$  is fixed at 0 or 1 for the current node, and a cut is generated, it is necessary to find a coefficient of  $x_t$  that makes the cut valid for the entire tree. Also, the typical valid cut generated for problem (*MIP*) before entering branch-and-bound does not have a nonzero coefficient for every variable. If variable  $x_t$  has a coefficient of zero in the cut, then it may be possible to strengthen the cut by finding a valid nonzero coefficient for variable  $x_t$ . The process of finding valid coefficients for binary variables fixed at 0 or 1; or, finding nonzero coefficients for variables in cuts with coefficient values of zero is called *lifting*. There is a vast literature on this topic. Work on the theoretical aspects of lifting includes Gomory [192], Padberg [366], Wolsey [459], Balas and Zemel [30], Zemel [473], and Nemhauser and Vance [354]. Crowder, Johnson, and Padberg [103] used lifting in their landmark computational study.

Suppose

$$\sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j \geq b \quad (15.38)$$

is a valid inequality for  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 0\}$  where  $t$  is an element of  $I \setminus K$  and as usual

$$\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \{0, 1\}, j \in I\}, \quad (15.39)$$

and  $C$  is an index set of the continuous variables. The procedure for taking the cut (15.38) and finding a coefficient  $a_t$  for variable  $x_t$  such that  $a_t x_t + \sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j \geq b$  is a valid cut for  $\Gamma$  is called *lifting*.

**Proposition 15.13** *If  $\sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j \geq b$  is a valid cut of  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 0\}$  for  $t \in I \setminus K$  and  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 1\} \neq \emptyset$ , then*

$$\sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j + (b - z_t)x_t \geq b \quad (15.40)$$

where  $b \leq z_t \leq z_t^*$  and

$$\begin{aligned} z_t^* := \min & \quad \sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j \\ \text{s.t.} & \quad x \in \Gamma \\ & \quad x_t = 1 \\ & \quad x_j \geq 0, \quad j = 1, \dots, n \\ & \quad x_j \in \{0, 1\}, \quad j \in I \end{aligned}$$

is a valid cut for  $\Gamma$ .

A analogous process is possible for when variables are fixed to one. Assume (15.38), is a valid cut for  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 1\}$ . It is possible to lift this cut as follows.

**Proposition 15.14** *If  $\sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j \geq b$  is a valid cut of  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 1\}$ , for  $t \in I \setminus K$  and  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 0\} \neq \emptyset$ , then*

$$\sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j + (z_t - b)x_t \geq z_t \quad (15.41)$$

where  $b \leq z_t \leq z_t^*$  and

$$\begin{aligned} z_t^* = \min & \sum_{j \in K} a_j x_j + \sum_{j \in C} a_j x_j \\ \text{s.t.} & x \in \Gamma \\ & x_t = 0 \\ & x_j \geq 0, \quad j = 1, \dots, n \\ & x_j \in \{0, 1\}, \quad j \in I \end{aligned}$$

is a valid cut for  $\Gamma$ .

Nemhauser and Wolsey [350] call the lifting of Proposition 15.13 *up-lifting* and the lifting of Proposition 15.14 *down-lifting*.

A very important aspect of lifting is that the lifted constraints (15.40) and (15.41) are at least as tight as the original constraint (15.38) since any point in  $\bar{\Gamma}$  that satisfies (15.40) or (15.41) also satisfies (15.38). If more than one variable in  $I \setminus K$  is fixed at zero or one, the lifting process can be done sequentially for each variable in  $I \setminus K$ , and is called *sequential lifting*. The order in which the variables are selected for lifting will affect their coefficients in the resulting lifted cut. See, for example, Zemel [473] for results on *simultaneous lifting*.

From a computational standpoint, it is not practical to solve an integer programming problem to find  $z_t^*$ . In practice it is typical to use only a subset of the constraints in  $\Gamma$  to generate  $z_t^*$ . For example, when generating minimal cover cuts, they may be lifted by solving an integer program using the constraint from which they were generated. This is illustrated next in a continuation of the generalized assignment problem example.

**Example 15.15 (Example 15.7 continued)** *In example 15.7 we generated the minimal cover inequality  $x_{11} + x_{31} + x_{41} \leq 2$  from the knapsack constraint*

$3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13$ . Or, after complementing by  $z_{ij} = 1 - x_{ij}$  and putting in  $\geq$  form, the valid inequality  $z_{11} + z_{31} + z_{41} \geq 1$  is generated from the knapsack constraint  $3z_{11} + 6z_{21} + 5z_{31} + 7z_{41} \geq 8$ . Perform down-lifting using Proposition 15.14 for variable  $z_{21}$  and solve the integer program

$$\begin{aligned} z_{21}^* &= \min && z_{11} + z_{31} + z_{41} \\ \text{s.t.} && 3z_{11} + 6z_{21} + 5z_{31} + 7z_{41} \geq 8 \\ && z_{21} = 0 \\ && z_{i1} \in \{0, 1\} \end{aligned}$$

The optimal solution value is  $z_{21}^* = 2$ . Then the lifted cut is  $z_{11} + z_{21} + z_{31} + z_{41} \geq 2$ . Adding this lifted cut to the other minimal cover cuts improves the objective function value of the linear programming relaxation of the generalized from 25.0 to 29.5. In this case finding the  $z_{21}^*$  requires solving a knapsack problem. By Proposition 15.14 having an optimal solution is not necessary. Martin and Schrage [318] solve the linear programming relaxation of the knapsack problem, then perform one branch and use the result for  $z_t$  in (15.41).

Recent computational studies of lifting within a branch-and-cut framework include Balas, Ceria, and Cornuéjols [27] and Gu, Nemhauser, and Savelsbergh [214]. Complexity issues associated with lifting are addressed in the companion paper Gu, Nemhauser, and Savelsbergh [215].

In the previous section on cutting planes, we stated that Gomory cuts were not practical for solving large integer programming problems. This was true in the 1950s and 1960s. Indeed, the poor results with Gomory cuts led to the development of linear programming based branch-and-bound. However, a very interesting recent study by Balas, Ceria, and Cornuéjols [26] demonstrates that incorporating Gomory cuts within a branch-and-cut framework is very valuable. They show that if the integer variables are 0/1 variables, Gomory cuts added at any node of the search tree are valid for the entire tree. By incorporating the Gomory cuts within a branch-and-cut algorithm as part of CPLEX they were able to solve more problem instances than using the CPLEX mixed integer solver without the Gomory cuts.

## 15.6 LAGRANGIAN CUTS

Again, consider problem (MIP),  $\min\{c^\top x \mid Ax \geq b, x \in \Gamma\}$ . In Chapter 12 we defined the Lagrangian function

$$L(u) = \min\{(c^\top - u^\top A)x + b^\top u \mid x \in \Gamma\}.$$

where  $\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathbb{Z}, j \in I\}$ . The proofs of the following propositions are left as Exercise 15.15.

**Proposition 15.16** *If  $\bar{u} \geq 0$ ,  $c^\top x + \bar{u}^\top(b - Ax) \geq L(\bar{u})$  is a valid cut for (MIP).*

If  $\bar{u}$  optimizes the Lagrangian dual, then it is possible to add a single cut that generates a bound equivalent to the convex hull relaxation with respect to dualizing the  $Ax \geq b$  constraints.

**Proposition 15.17** *If  $\bar{u} \geq 0$  is an optimal vector of Lagrangian dual multipliers, then the optimal solution value of  $\min \{c^\top x \mid Ax \geq b, x \in \text{conv}(\Gamma)\}$  is equal to the optimal solution value of the linear programming relaxation of (MIP) with the single added cut*

$$c^\top x + \bar{u}^\top(b - Ax) \geq L(\bar{u}).$$

At this point, it is instructive to compare the Lagrangian cuts with polyhedral based cuts such as the minimal cover cuts. Most integer programs are very sparse, which means given constraints  $(a^i)^\top x \geq b_i$  and  $(a^j)^\top x \geq b_j$  there is little overlap of variables with nonzero coefficients. As explained earlier, the idea behind polyhedral based cuts such as minimal cover cuts is to treat each constraint individually (or consider small subsets of constraints with special structure) and generate as closely as possible the convex hull of each  $\Gamma_i = \{x \in \mathbb{R}^n \mid (a^i)^\top x \geq b_i, x \geq 0, x \in \mathbb{Z}^n\}$ . If the problem is sparse,  $\text{conv}(\cap_{i=1}^m \Gamma_i)$  should closely approximate  $\cap_{i=1}^m \text{conv}(\Gamma_i)$  and we end up with a tight formulation after adding the cuts. One could do the same thing with Lagrangian cuts, and sequentially optimize each of the Lagrangian dual problems corresponding to  $\Gamma_i$ . By Proposition 15.17, adding each of these Lagrangian cuts gives  $\cap_{i=1}^m \text{conv}(\Gamma_i)$ . This idea is called *Lagrangian decomposition*. See Guignard and Kim [216]. The problem is that the Lagrangian cuts will be very dense and generating each of them may be rather time consuming.

## 15.7 INTEGER PROGRAMMING TEST PROBLEMS

There are a number of software packages available for solving mixed integer linear optimization problems. These packages implement most of the ideas,

including preprocessing and cut generation, discussed in this chapter. Commercial packages include CPLEX, LINDO, and OSL. Researchers Nemhauser, Savelsbergh, and Sigismondi [353] at the Georgia Institute of Technology developed MINTO, which is a linear programming based branch-and-bound algorithm designed to work with subroutines available in OSL and CPLEX. For a more comprehensive list of mixed integer optimization software visit the following WEB site:

<http://www-c.mcs.anl.gov/home/otc/Guide/SoftwareGuide/>

The information in this WEB site is taken from the *Optimization Software Guide* by Moré and Wright [346]. Comparing mixed integer optimization codes is much like comparing word processors or spreadsheets. They each have their own unique features depending upon the particular release. When one package adds a useful feature the others tend to do so also, and enhance it, in the next release.

An important aspect of recent rapid improvement in integer programming methodology is the widespread availability of large, realistic problems. This enables researchers to systematically reproduce results and make valid comparisons. A large set of mixed integer linear optimization problems with different structures is available at the following WEB site.

<http://www.caam.rice.edu/~bixby/miplib/miplib3.html>

A large set of lot sizing and scheduling algorithms is available at

<ftp://gsbkip.uchicago.edu/Files/faculty/martin/iptest/lotsize/>

## 15.8 CONCLUSION

The first algorithms for solving integer programs were cutting plane algorithms. These were developed in the 1950s and generally were not capable of solving integer programs arising from real applications. See, for example, Gomory [191]. Linear programming based branch-and-bound was the first “practical” algorithm for solving integer programming problems. In the early 1970s there

was a move away from linear programming based branch-and-bound methods and research focused on using the Lagrangian dual as another means of generating improved bounds on the optimal integer programming solution value. Then, in 1983 Crowder, Johnson, and Padberg [103] published their landmark paper. The procedure described in their paper used many of the preprocessing methods described in Section 15.2, the cuts of Section 15.3, and lifting methods of Section 15.5. They were able to solve integer programs which could not be solved by standard linear programming based branch-and-bound algorithms. Shortly thereafter, the branch-and-cut algorithm described in Section 15.4 became popular. Cuts, preprocessing, lifting, and branch-and-cut are the focus of much of the current research in integer programming.

## 15.9 EXERCISES

- 15.1 Let  $\Gamma = \{x \mid \sum_{j \in I} a_j x_j \geq b, x_j \in \{0, 1\}, j \in I\}$ . Given  $h > 0$ , prove that a valid cut of  $\Gamma$  is given by

$$\sum_{j \in I} \lceil a_j/h \rceil x_j \geq \lceil b/h \rceil.$$

- 15.2 Show that  $x_1 + x_2 + 2x_3 \geq 2$  is a valid cut of

$$\Gamma = \{(x_1, x_2, x_3) \mid 21.5x_1 + 43x_2 + 86x_3 \geq 45, x_1, x_2, x_3 \in \{0, 1\}\}.$$

- 15.3 Prove Lemma 15.6.

- 15.4 Show that the disaggregated constraints  $x_{ij} \leq y_i$  for the simple plant location model (*SPL*) are minimal cover inequalities for the aggregated constraint  $\sum_{j=1}^m x_{ij} \leq my_i$  used in formulation (*SPLAGG*).

- 15.5 Show how to generate all of the  $(l, S)$  cuts by repeated application of CACR.

- 15.6 It is possible to generate all of the disaggregated constraints  $x_{ij} \leq y_i$  for the simple plant location model (*SPL*) by applying CACR to the aggregated model (*SPLAGG*). Assume a solution  $(\bar{x}, \bar{y})$  to the linear programming relaxation of (*SPLAGG*). Show how to find a violated disaggregated cut (if one exists) by solving a knapsack separation problem. This separation problem is a specialization of (15.34)-(15.35) to simple plant location.

- 15.7 Show that the cut (15.28) is valid as long as  $\lambda \leq b_i - \sum_{j \in W} a_{ij}$ .

- 15.8 Code in C or FORTRAN a subroutine which takes the system  $Ax \geq b$  with  $A$  and  $b$  rational, as input and converts it to an equivalent system  $A'x \geq b'$  where  $A'$  is an integer matrix. Assume  $A$  is stored in column form.
- 15.9 Calculate valid upper bounds on the inventory variables in the dynamic lot size model of Example 15.3. If the demand constraint in the last time period is written as  $I_{i4} + x_{i5} - I_{i5} = d_{i5}$ , what effect does this have on the calculation of upper and lower bounds?
- 15.10 Consider the demand constraints  $I_{t-1} + x_t - I_t = d_t$  (we have dropped the product index) and nonnegativity constraints  $I_t \geq 0$  in the dynamic lot size model. Project out the inventory variables. Assuming  $T$  time periods how many constraints result from projection and what is the nonzero density of the resulting system?
- 15.11 Code in C or FORTRAN a subroutine which takes as input an  $m \times n$  matrix  $A$  stored in column form and outputs a function  $f$  with domain  $D = \{1, 2, \dots, m\}$  such that for all  $i_1, i_2 \in D$ ,  $f(i_1) = f(i_2)$  if and only if row  $i_1$  is a multiple of row  $i_2$ .
- 15.12 If a preprocessing algorithm discovers that dual constraint  $j$  is redundant, what does this imply about primal variable  $j$ ?
- 15.13 Prove that if  $\Gamma_t = \Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 0\}$  is full dimensional and (15.38) is a facet for  $\text{conv}(\Gamma_t)$  and  $\Gamma \cap \{x \in \mathbb{R}^n \mid x_t = 1\} \neq \emptyset$ , then  $\Gamma$  is full dimensional and (15.40) is a facet of  $\text{conv}(\Gamma)$ .
- 15.14 Apply Proposition 15.13 to the inequality  $x_{11} + x_{31} + x_{41} \leq 2$  in Example 15.7.
- 15.15 Prove Propositions 15.16 and 15.17.
- 15.16 Generate a Lagrangian cut for the generalized assignment problem of Example 15.7 based on the Lagrangian dual which results from dualizing the assignment constraints.
- 15.17 Generate a Lagrangian cut for the dynamic lot sizing model of Example 15.3 based on the Lagrangian dual which results from dualizing the capacity constraints.

---

# LARGE INTEGER PROGRAMS: PROJECTION AND INVERSE PROJECTION

## 16.1 INTRODUCTION

In this chapter we continue with the theme of generating better mixed integer model formulations. The mixed integer linear program under consideration is

$$\begin{aligned} (MIP) \quad & \min c^\top x \\ & \text{s.t. } Ax \geq b \\ & \quad Bx \geq d \\ & \quad x \geq 0 \\ & \quad x_j \in \mathbb{Z}, j \in I \end{aligned}$$

where  $x \in \mathbb{R}^n$  and  $I$  is an index set contained in  $\{1, \dots, n\}$ . This mixed integer linear program can, in theory, be solved using linear programming based branch-and-bound. In Chapter 15 we saw that the success of a linear programming based branch-and-bound algorithm depended upon having a good or tight linear programming relaxation. In that chapter, the formulation  $(MIP)$  was strengthened by identifying a subset of the problem constraints

$$\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathbb{Z}, j \in I\}$$

with special structure and then using the special structure to fix variables, perform coefficient reduction and generate cutting planes. These reformulation techniques take as a given, the variable definition used to represent the “real world problem” as  $(MIP)$ . However, for a given problem there may be alternative variable definitions, some better than others in terms of generating good linear programming relaxations. In this chapter, we develop a philosophy of problem reformulation based on generating alternative formulations which use *auxiliary variables*. Consider the following examples.

**Example 16.1** Define a polytope in  $\mathbb{R}^2$  by  $\Gamma = \{(x_1, x_2) \mid |x_1| + |x_2| \leq 1\}$ . The linear inequalities that define this polytope are  $x_1 + x_2 \leq 1$ ,  $x_1 - x_2 \leq 1$ ,  $-x_1 + x_2 \leq 1$  and  $-x_1 - x_2 \leq 1$ . However, another representation of this polytope is

$$\begin{aligned} z_1^+ - z_1^- &= x_1 \\ z_2^+ - z_2^- &= x_2 \\ z_1^+ + z_1^- + z_2^+ + z_2^- &\leq 1 \\ z_1^+, z_1^-, z_2^+, z_2^- &\geq 0. \end{aligned}$$

Projecting out the additional auxiliary variables  $z_1^+, z_1^-, z_2^+$  and  $z_2^-$  results in the inequalities that define  $\Gamma$ .

In this example  $\Gamma$  was a polytope. However,  $\Gamma$  might be the feasible region of a mixed integer linear program. Let  $\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathcal{Z}, j \in I\}$ . Recall the mixed integer finite basis theorem from Chapter 4. Assume all data are rational.

**Theorem 16.2 (Mixed Integer Finite Basis Theorem)** If  $\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathcal{Z}, j \in I\}$ , then  $\text{conv}(\Gamma)$  is a polyhedron. Then there exist  $x^1, \dots, x^r \in \Gamma$  such that

$$\begin{aligned} \text{conv}(\Gamma) &= \{x \in \mathbb{R}^n \mid x = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i, \\ &\quad \sum_{i=1}^q z_i = 1, z_i \geq 0, i = 1, \dots, r\} \end{aligned} \tag{16.1}$$

This result implies that there are at least two representations of  $\text{conv}(\Gamma)$ . One representation is given by (16.1) in the  $x$  variables plus auxiliary  $z$  variables. However,  $\text{conv}(\Gamma)$  is also a polyhedron in  $\mathbb{R}^n$  and therefore has a representation as linear inequalities in only the original  $x$  variables. The representation of  $\text{conv}(\Gamma)$  using auxiliary variables yields the following alternative formulation of (MIP).

$$\begin{aligned} \min & e^\top x \\ (\text{MIPFB}) \quad & \text{s.t.} & Ax \geq b \\ & \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i = x \end{aligned}$$

$$\begin{aligned} \sum_{i=1}^q z_i &= 1 \\ z_i &\geq 0, \quad i = 1, \dots, r \\ x_j &\in \mathbb{Z}, \quad j \in I \end{aligned}$$

In formulation  $(MIPFB)$ , the  $x^i$  for  $i = q + 1, \dots, r$  are rays of the recession cone of  $\text{conv}(\Gamma)$ . The significance of this reformulation is that the optimal solution value of the linear programming relaxation of  $(MIPFB)$  is equal to the optimal solution value of the convex hull relaxation of  $(MIP)$ . Recall from Chapter 12, that the convex hull relaxation of  $(MIP)$  is

$$\begin{aligned} (\text{conv}(MIP)) \quad \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq b \\ & x \in \text{conv}(\Gamma) \end{aligned}$$

and that the optimal solution value of  $(\text{conv}(MIP))$  is equal to the optimal value of the Lagrangian dual of  $(MIP)$  when the  $Ax \geq b$  constraints are dualized. Thus, solving the linear programming relaxation of formulation  $(MIPFB)$ , with auxiliary variables, is another way to generate bounds equivalent to the Lagrangian dual. In Chapter 12 we saw that one way to optimize  $(\text{conv}(MIP))$  was by using column generation. Later in this chapter we apply column generation to formulation  $(MIPFB)$ .

**Example 16.3** *The integer program is*

$$\begin{aligned} \min \quad & -x_1 - x_2 \\ (MIP) \quad \text{s.t.} \quad & x_1 - x_2 \leq 2 \\ & (x_1, x_2) \in \Gamma \end{aligned}$$

where  $\Gamma$  is defined by

$$\Gamma = \{(x_1, x_2) \mid 4x_1 + 9x_2 \leq 18, -2x_1 + 4x_2 \leq 4, x_1, x_2 \in \mathbb{Z}_+\}.$$

Clearly  $\text{conv}(\Gamma)$  is a polytope and is generated by taking the convex hull of the four points  $(0, 0)$ ,  $(4, 0)$ ,  $(2, 1)$  and  $(0, 1)$ . An alternative formulation of  $(MIP)$  using the mixed integer finite basis theorem is

$$\begin{aligned} \min \quad & -x_1 - x_2 \\ \text{s.t.} \quad & x_1 - x_2 \leq 2 \\ & 4z_2 + 2z_3 = x_1 \end{aligned}$$

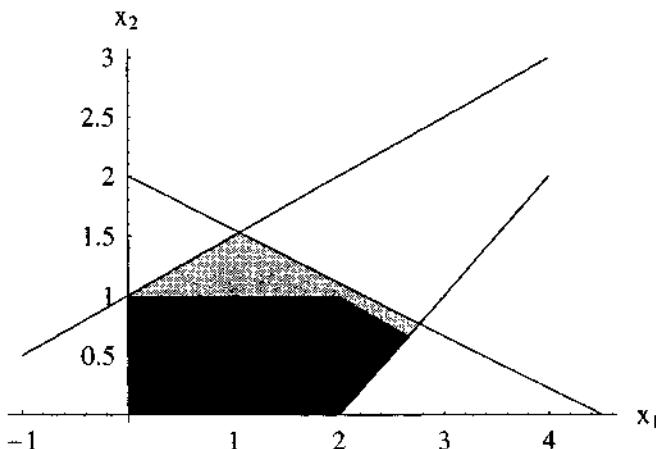
$$\begin{aligned}
 z_3 + z_4 &= x_2 \\
 z_1 + z_2 + z_3 + z_4 &= 1 \\
 z_i &\geq 0, \quad i = 1, \dots, 4 \\
 x_1, x_2 &\in \mathbb{Z}_+
 \end{aligned}$$

If the auxiliary  $z$  variables are projected out, the resulting system of nonredundant constraints in the original  $(x_1, x_2)$  variables is

$$\begin{aligned}
 x_1 - x_2 &\leq 2 \\
 (1/2)x_1 + x_2 &\leq 2 \\
 x_2 &\leq 1 \\
 x_1, x_2 &\in \mathbb{Z}_+
 \end{aligned}$$

In Figure 16.1, the feasible region of the linear relaxation of this system is depicted by the darkly shaded polytope. The feasible region in  $(x_1, x_2)$  space of the linear relaxation of the original formulation is the darkly shaded polytope plus the lightly shaded region. The feasible region of the linear programming relaxation of the original formulation strictly contains the feasible region of the linear programming relaxation of the alternative formulation. The addition of the auxiliary variables has tightened the formulation. See also the discussion of this example as Example 12.7 in Chapter 12.

**Figure 16.1** Projection of Auxiliary Variable Formulation



**Example 16.4** Refer to the traveling salesman problem formulation in Subsection 1.3.3 in Chapter 1. In this example assume that the truck must visit all  $n$  cities and city 0 is the depot city.

$$\begin{aligned}
 \min \quad & \sum_{i=0}^n \sum_{\substack{j=0 \\ i \neq j}}^n c_{ij} x_{ij} \\
 (TSP) \quad \text{s.t.} \quad & \sum_{\substack{i=0 \\ i \neq j}}^n x_{ij} = 1, \quad j = 0, \dots, n \\
 & \sum_{\substack{j=0 \\ i \neq j}}^n x_{ij} = 1, \quad i = 0, \dots, n \\
 & \sum_{\substack{(i,j) \in S \times S \\ i \neq j}} x_{ij} \leq |S| - 1, \quad S \subseteq \{1, \dots, n\}, \quad 2 \leq |S| \leq n - 1 \quad (16.2) \\
 & x_{ij} \in \{0, 1\}, \quad i \neq j
 \end{aligned}$$

In this formulation  $x_{ij} = 1$  if arc  $(i, j)$  is in the tour, that is, city  $j$  is visited immediately after city  $i$ . As explained in Chapter 1, the key to this formulation is the set of subtour breaking constraints (16.2) due to Dantzig, Fulkerson, and Johnson [114]. Consider an alternative variable formulation due to Flood [151] where  $z_{ijk} = 1$  if the  $k$ th arc on the tour is from city  $i$  to  $j$ . The reformulation using the new variables is

$$\begin{aligned}
 \min \quad & \sum_{i=0}^n \sum_{\substack{j=0 \\ j \neq i}}^n c_{ij} x_{ij} \\
 (RTSP) \quad \text{s.t.} \quad & \sum_{\substack{i=0 \\ i \neq j}}^n x_{ij} = 1, \quad j = 0, \dots, n \\
 & \sum_{\substack{j=0 \\ i \neq j}}^n x_{ij} = 1, \quad i = 0, \dots, n \\
 & \sum_{j=1}^n z_{0j1} = 1 \\
 & \sum_{i=0}^n z_{ijk} - \sum_{l=0}^n z_{j,l,k+1} = 0, \quad j = 1, \dots, n, \quad k = 1, \dots, n \\
 & \sum_{k=1}^{n+1} z_{ijk} = x_{ij}, \quad i \neq j
 \end{aligned}$$

$$z_{ijk} \in \{0, 1\}, \quad k = 1, \dots, n+1, \quad i \neq j.$$

These examples illustrate that alternative formulations using *different variable definitions* exist for the same problem. Perhaps the earliest author to discuss alternative variable formulations for integer programming was Dantzig in his book, *Linear Programming and Extensions*, where he gives three formulations for the traveling salesman problem using different variable definitions. The two formulations in Example 16.4 are from the Dantzig book. See Gouveia [204] for a comparison of the linear programming relaxation of several different traveling salesman formulations. In this chapter, we develop methods for generating auxiliary variables for reformulating models in order to improve the linear programming relaxations.

In problem  $(MIPFB)$  there is a one to one correspondence between the auxiliary variables and the generators of  $\text{conv}(\Gamma)$ . However, other auxiliary variable formulations of problem  $(MIP)$  may exist. Let  $\Gamma$  denote an arbitrary subset of  $\mathbb{R}^n$ . If

$$\text{conv}(\Gamma) = \{x \mid \text{there is a } z \text{ such that } Fx + Gz \geq h\}$$

then,  $\text{conv}(\Gamma)$  is a polyhedron and we say that the system  $Fx + Gz \geq h$  is an *extended polyhedral representation* of  $\text{conv}(\Gamma)$  with auxiliary variables  $z$ .

If  $\Gamma \subseteq \mathbb{R}^n$  and the number of variables and constraints in the extended formulation is a polynomial function of  $n$  we say it is a *compact* extended formulation. In general,  $(MIPFB)$  is not a compact reformulation since the number of generators of  $\text{conv}(\Gamma)$  is an exponential function of the number of variables and constraints that define  $\Gamma$ .

Given an extended polyhedral representation of  $\text{conv}(\Gamma)$  where  $\Gamma = \{x \in \mathbb{R}^n \mid Bx \geq d, x \geq 0, x_j \in \mathcal{Z}, j \in I\}$ , the corresponding reformulation of  $(MIP)$  using auxiliary variables is

$$\min c^\top x \tag{16.3}$$

$$(RMIP) \quad \text{s.t.} \quad Ax \geq b \tag{16.4}$$

$$Fx + Gz \geq h \tag{16.5}$$

$$x_j \in \mathcal{Z}, \quad j \in I. \tag{16.6}$$

If  $Fx + Gz \geq h$  is an extended polyhedral representation of  $\text{conv}(\Gamma)$ , the optimal solution value of the linear programming relaxation of  $(RMIP)$  is equal to the optimal solution value of  $\text{conv}(MIP)$ , which is equal to the optimal value of the Lagrangian dual. In the proposition below we give a necessary and sufficient condition for  $Fx + Gz \geq h$  to be an extended polyhedral representation of  $\text{conv}(\Gamma)$ .

**Proposition 16.5** If  $\Gamma \subseteq \mathbb{R}^n$  and  $\text{conv}(\Gamma)$  is a closed set, the system  $Fx + Gz \geq h$  is an extended polyhedral representation of  $\text{conv}(\Gamma)$  if and only if  $\min \{c^\top x \mid x \in \text{conv}(\Gamma)\} = \min \{c^\top x \mid Fx + Gz \geq h\}$  for all  $c \in \mathbb{R}^n$ .

If the inequalities  $Fx + Gz \geq h$  are an extended polyhedral representation of  $\text{conv}(\Gamma)$  and they define an integer polyhedron, then  $\text{conv}(\Gamma)$  is an integer polyhedron. However, if  $\text{conv}(\Gamma)$  is an integer polyhedron, and  $Fx + Gz \geq h$  is an extended polyhedral representation of  $\text{conv}(\Gamma)$ , it does not follow that the inequalities  $Fx + Gz \geq h$  define an integer polyhedron.

**Example 16.6 (Special Ordered Sets (Tomlin [430]))** Let  $\Gamma = \cup_{i=1}^{n-1} \Gamma_i$  where

$$\begin{aligned}\Gamma_i = \{(x_1, \dots, x_n) \mid &x_i + x_{i+1} = 1, x_i, x_{i+1} \geq 0, \\ &x_k = 0, k = 1, \dots, n, k \neq i, k \neq i+1\}.\end{aligned}$$

The  $\Gamma_i$  are special ordered sets of the second type and are useful in “linearizing” nonlinear programs. An extended polyhedral representation of  $\text{conv}(\Gamma)$  is generated by introducing an auxiliary variable  $z_i$  which is positive if  $x \in \Gamma_i$ .

$$\sum_{i=1}^n x_i = 1, \quad \sum_{i=1}^{n-1} z_i = 1 \tag{16.7}$$

$$x_1 \leq z_1 \tag{16.8}$$

$$x_k \leq z_{k-1} + z_k, \quad k = 2, \dots, n-1 \tag{16.9}$$

$$x_n \leq z_{n-1} \tag{16.10}$$

$$x, z \geq 0 \tag{16.11}$$

To see that (16.7)-(16.11) is an extended polyhedral representation of  $\text{conv}(\Gamma)$ , observe that an optimal solution to  $\min \{c^\top x \mid (16.7) - (16.11)\}$  is  $\bar{x}_l = 1$ ,  $\bar{z}_{l-1} = 1$  when  $l > 1$  and  $\bar{z}_l = 1$  when  $l = 1$  where  $l = \arg\min \{c_i \mid i = 1, \dots, n\}$ . Therefore,  $\min \{c^\top x \mid x \in \text{conv}(\Gamma)\} = \min \{c^\top x \mid (16.7) - (16.11)\}$  for all  $c \in \mathbb{R}^n$  so (16.7)-(16.11) is an extended polyhedral representation of  $\text{conv}(\Gamma)$  by Proposition 16.5. Although  $\text{conv}(\Gamma)$  is an integer polyhedron, it is left as Exercise 16.13 to show that the polyhedron defined by (16.7)-(16.11) is not an integer polyhedron.

This example also illustrates the major theme of this chapter that the same problem may have alternative formulations involving different sets of variables. Refer back to equations (9.3)-(9.5) in Chapter 9. Which formulation is better?

Proposition 16.5 provides a method for proving that a formulation with auxiliary variables is an extended polyhedral formulation of a polyhedron  $\text{conv}(\Gamma)$ . The proof technique was illustrated in Example 16.6. The following is another proposition used in proving that auxiliary variable reformulations are valid extended polyhedral formulations.

**Proposition 16.7** *The system  $F\bar{x} + Gz \geq h$  is an extended polyhedral representation of the polyhedron  $\text{conv}(\Gamma)$  if and only if*

1.  $F\bar{x} + Gz \geq h$ , then  $\bar{x} \in \text{conv}(\Gamma)$ ;
2. given a finite set  $x^1, \dots, x^q$  of generators of  $\text{conv}(\Gamma)$ , there exist  $z^1, \dots, z^q$  such that  $Fx^i + Gz^i \geq h$ ,  $i = 1, \dots, q$ .

Proposition 16.7 is really just another way to state the projection of  $Fx + Gz \geq h$  into the  $x$  variable space is  $\text{conv}(\Gamma)$ . Condition 1 states that in the reformulation with auxiliary variables we have not *created any new solutions*. Condition 2 states that in the reformulation with auxiliary variables we have not *destroyed any solutions*. Proposition 16.7 is used in the next section.

Even if  $Fx + Gz \geq h$  is not an extended polyhedral representation of  $\text{conv}(\Gamma)$  it is stronger or tighter than the linear relaxation of  $\Gamma$  if

$$\{x \mid \text{there is a } z \text{ such that } Fx + Gz \geq h\} \subset \bar{\Gamma}.$$

In Section 16.2 of this chapter we examine four methods for reformulating (MIP) using auxiliary variables. In these four methods the auxiliary variables no longer correspond to the generators of  $\text{conv}(\Gamma)$ . We provide several examples of compact reformulations. In Section 16.3 we address the problem of finding the minimum size reformulation in auxiliary variables. In many applications, the reformulated model is not compact (for example (MIPFB) where there is an auxiliary variable for every extreme point and extreme ray) and has too many auxiliary variables to solve explicitly. In this case, the Dantzig-Wolfe (column generation) and Benders' (projection) decomposition methods developed earlier apply. In Section 16.4 we apply column generation to the extended formulations within a branch-and-bound context. In Section 16.5 we apply the Benders' decomposition algorithm to extended formulations and project out the auxiliary variables. This yields a cutting plane algorithm where the Benders' relaxed master corresponds to the linear integer program in the original variables. Concluding and summary comments are made in Section 16.6. Exercises are provided in Section 16.7.

Throughout the book we have given a number of examples where subsets  $\Gamma \subseteq \mathbb{R}^n$  are modeled as integer linear programs. However, we do not address the more fundamental question of characterizing the subsets  $\Gamma \subseteq \mathbb{R}^n$  that can be modeled as integer programs. Meyer [334, 333], Ibaraki [239], Jeroslow [248], and Jeroslow and Lowe [249] all addressed this problem. Jeroslow and Lowe provide a necessary and sufficient characterization. In the next section we give an example of an extended polyhedral representation of  $\text{conv}(\Gamma)$ , yet  $\Gamma$  cannot be modeled as a mixed integer program. Jeroslow and Lowe [249] call extended polyhedral representations *sharp*.

## 16.2 AUXILIARY VARIABLE METHODS

The auxiliary variables in model (*MIPFB*) are derived from the generators of  $\text{conv}(\Gamma)$ . In this section we look at four other methods for generating auxiliary variables to tighten the linear programming relaxation of (*MIP*).

### 16.2.1 Network Flow Reformulations

Many problems involving location, distribution and production scheduling can be formulated as a fixed charge network flow problem. Perhaps the first authors to use extended formulations with auxiliary variables in order to tighten the linear programming relaxation were Rardin and Choe [376] who gave two reformulations for the uncapacitated network flow problem. Below is the generic formulation (*UNF*), for the single source minimum cost network flow problem with fixed charges on the arcs. In this formulation,  $\mathcal{A}$  is an index set of directed arcs,  $V$  is an index set of vertices, vertex 0 is the supply vertex,  $D$  indexes the demand vertices where  $d_j$  is the nonnegative demand at vertex  $j$ ,  $c_{ij}$  is the marginal cost of a unit of flow on arc  $(i, j)$ , and  $f_{ij}$  is the fixed cost associated with positive flow on arc  $(i, j)$ . The variables are  $x_{ij}$ , the flow on arc  $(i, j)$  and  $y_{ij}$ , a binary variable which is 1 if there is positive flow on arc  $(i, j)$  and 0 otherwise.

$$\min \sum_{(i,j) \in \mathcal{A}} c_{ij}x_{ij} + \sum_{(i,j) \in \mathcal{A}} f_{ij}y_{ij} \quad (16.12)$$

$$(UNF) \quad \text{s.t.} \quad \sum_{(j,i) \in \mathcal{A}} x_{ji} - \sum_{(l,j) \in \mathcal{A}} x_{lj} = \begin{cases} -\sum_{i \in D} d_i, & l = 0 \\ d_l, & l \in D \\ 0, & l \notin D \cup \{0\} \end{cases} \quad (16.13)$$

$$x_{ij} \leq (\sum_{l \in D} d_l)y_{ij}, \quad (i, j) \in \mathcal{A} \quad (16.14)$$

$$x, y \geq 0 \quad (16.15)$$

$$y_{ij} \in \{0, 1\}, \quad (i, j) \in \mathcal{A} \quad (16.16)$$

Let  $\Gamma = \{(x, y) \mid (x, y) \text{ satisfies (16.13)} - \text{(16.16)}\}$ . Capacity constraints, if necessary, are added to the model by changing the coefficient of  $y_{ij}$  in the fixed charge constraint (16.14) to the capacity of arc  $(i, j)$ . Model (UNF) is an example of the minimum cost network flow (MCNF) first presented in Section 14.1 in Chapter 14 with fixed charges on the arcs (constraints (16.14)) and only one source.

The first reformulation for (UNF) is the *multicommodity reformulation*. The multicommodity flow reformulation is a disaggregated formulation. The key idea is to create a distinct commodity for each demand vertex. The flow on each arc  $(i, j)$  is disaggregated into the flow of each commodity on arc  $(i, j)$ . For each arc-demand combination there is an auxiliary variable  $z_{ijk}$  which represents the flow of commodity  $k$  on arc  $(i, j)$ . The total flow on arc  $(i, j)$  is the sum of the commodity flows on arc  $(i, j)$  and this is expressed in the constraint  $\sum_{k \in D} z_{ijk} = x_{ij}$ . The new variable definition allows for the fixed charge constraint  $x_{ij} \leq (\sum_{l \in D} d_l)y_{ij}$  to be disaggregated into  $z_{ijk} \leq d_k y_{ij}$ . The reformulated model with auxiliary variables is given below.

$$\begin{aligned} \min \quad & \sum_{(i, j) \in \mathcal{A}} c_{ij}x_{ij} + \sum_{(i, j) \in \mathcal{A}} f_{ij}y_{ij} \\ (MCUNF) \text{ s.t.} \quad & \sum_{(i, l) \in \mathcal{A}} z_{ilk} - \sum_{(l, i) \in \mathcal{A}} z_{lik} = \left\{ \begin{array}{ll} -d_k, & l = 0 \\ 0, & l \neq 0, k \\ d_k, & l = k \end{array} \right\}, \quad k \in D \\ & z_{ijk} \leq d_k y_{ij}, \quad (i, j) \in \mathcal{A}, \quad k \in D \\ & \sum_{k \in D} z_{ijk} = x_{ij}, \quad (i, j) \in \mathcal{A} \\ & x, y, z \geq 0 \\ & y_{ij} \in \{0, 1\}, \quad (i, j) \in \mathcal{A} \end{aligned}$$

The intuition for why this formulation is effective is the following. If the flow of commodity  $k$  on arc  $(i, j)$  is  $d_k$  then the fixed charge constraint  $z_{ijk} \leq d_k y_{ij}$  forces the integer variable  $y_{ij}$  to 1. At worst, in any linear programming relaxation  $(\bar{x}, \bar{y}, \bar{z})$  of (MCUNF) the value of  $\bar{y}_{ij}$  is at least as large as the fraction of demand for commodity  $k$  flowing through arc  $(i, j)$ . That is,

$$\bar{y}_{ij} = \max\{\bar{z}_{ijk} \mid k \in D\}.$$

This is not the case for the fixed charge constraints (16.14) in model (UNF) where, for example, 100 per cent of demand for commodity  $k$  can flow through

arc  $(i, j)$  yet  $\bar{y}_{ij}$  is a small positive number if no other commodities use arc  $(i, j)$ .

Unfortunately, extreme point solutions of the linear relaxation of  $(MCUNF)$  are not always binary in the  $y$  variables and it is not an extended polyhedral representation of  $(UNF)$ . However, its linear relaxation is never worse, and usually much better, than the linear relaxation of  $(UNF)$ .

**Proposition 16.8** *If  $(\bar{x}, \bar{y}, \bar{z})$  is a feasible solution to the linear programming relaxation of  $(MCUNF)$ , then  $(\bar{x}, \bar{y})$  is a feasible solution to the linear programming relaxation of  $(UNF)$  and  $z_0(\overline{MCUNF}) \geq z_0(\overline{UNF})$ .*

**Proof:** If  $(\bar{x}, \bar{y}, \bar{z})$  is a feasible solution to the linear programming relaxation of  $(MCUNF)$ , then  $\bar{z}_{ijk} \leq d_k \bar{y}_{ij}$  and adding over commodities gives

$$\bar{x}_{ij} = \sum_{k \in D} \bar{z}_{ijk} \leq \left( \sum_{k \in D} d_k \right) \bar{y}_{ij}.$$

Aggregating the flow balance constraints of  $(MCUNF)$  in a similar fashion yields the flow balance constraints for model  $(UNF)$  and  $(\bar{x}, \bar{y})$  is feasible to  $(UNF)$ . Since both formulations have the same objective function,  $z_0(\overline{MCUNF}) \geq z_0(\overline{UNF})$ .  $\square$

**Example 16.9** *In Example 15.11 in Chapter 15 we developed cuts for the single product dynamic lot size mixed integer program (MPDLS). The formulation for this problem is*

$$\min \sum_{t=1}^T (c_t x_t + h_t I_t + f_t y_t) \quad (16.17)$$

$$(DLS) \quad \text{s.t.} \quad I_{t-1} + x_t - I_t = d_t, \quad t = 1, \dots, T \quad (16.18)$$

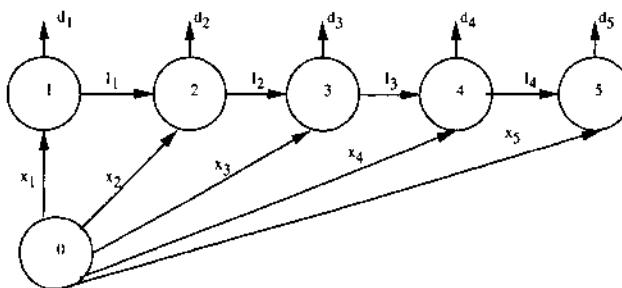
$$x_t \leq \tau_{tT} y_t, \quad t = 1, \dots, T \quad (16.19)$$

$$x_t, I_t \geq 0, \quad t = 1, \dots, T \quad (16.20)$$

$$I_0, I_T = 0 \quad (16.21)$$

$$y_t \in \{0, 1\}, \quad t = 1, \dots, T. \quad (16.22)$$

The parameters are  $T$ , the number of time periods in the planning horizon;  $d_t$ , the demand in period  $t$ ;  $\tau_{tk} = \sum_{j=t}^k d_j$ , the sum of demand in periods  $t$  through  $k$ ;  $f_t$ , the fixed cost associated with production in period  $t$ ;  $h_t$ , the marginal cost

**Figure 16.2** A Five Period Dynamic Lot Size Model

of holding one unit in inventory at the end of period  $t$ ; and  $c_t$ , the marginal production cost of one unit in period  $t$ .

This version of the problem is really uncapacitated, since in any time period  $t$ , production will never exceed  $\tau_{tT}$ .

The variables are  $x_t$ , the units of production in period  $t$ ;  $I_t$ , the units of inventory at the end of period  $t$ ; and  $y_t$ , a binary variable which is fixed to 1 if there is nonzero production in period  $t$ , otherwise it is fixed to 0. In Figure 16.2 we illustrate the network for a five period problem. There are five demand nodes, 1, 2, 3, 4 and 5 respectively, and one supply node 0. The variable  $x_t$  corresponds to flow from supply vertex 0 to demand vertex  $t$  and variable  $I_t$  corresponds to flow from vertex  $t$  to vertex  $t + 1$ . Conservation of flow constraints (16.13) in the uncapacitated network flow model (UNF) correspond to the demand and inventory balance constraints (16.18) in the dynamic lot size model (DLS).

Below is the complete formulation using the multicommodity flow auxiliary variables for a five period example. We use the cost and demand data for the first product of Example 12.20. In the reformulation below the variable  $z_{ij}$  is the flow of commodity  $j$  from the supply vertex 0 to vertex  $i$ . Think of  $z_{ij}$  as the quantity of period  $j$  demand satisfied from production in period  $i$ . The flow of commodity  $j$  from vertex  $i$  to  $i + 1$  for  $i = 1, \dots, 4$  is  $w_{ij}$ . Think of  $w_{ij}$  as the amount of period  $j$  demand held in inventory at the end of period  $i$ . In the formulation below constraints 2)-16) are the flow conservation constraints at each vertex for each commodity.

$$\begin{aligned} \text{MIN } & 150 Y_1 + 140 Y_2 + 160 Y_3 + 160 Y_4 + 160 Y_5 + 7 X_1 + 7 X_2 + 8 X_3 \\ & + 7 X_4 + 7 X_5 + I_1 + I_2 + 2 I_3 + 2 I_4 \\ \text{SUBJECT TO } & \end{aligned}$$

```

2) Z11 =    60
3) Z12 - W12 =    0
4) Z13 - W13 =    0
5) Z14 - W14 =    0
6) Z15 - W15 =    0
7) W12 + Z22 =   100
8) W13 + Z23 - W23 =    0
9) W14 + Z24 - W24 =    0
10) W15 + Z25 - W25 =    0
11) W23 + Z33 =   140
12) W24 + Z34 - W34 =    0
13) W25 + Z35 - W35 =    0
14) W34 + Z44 =   200
15) W35 + Z45 - W45 =    0
16) W45 + Z55 =   120
17) - 60 Y1 + Z11 <=  0
18) - 100 Y1 + Z12 <=  0
19) - 140 Y1 + Z13 <=  0
20) - 200 Y1 + Z14 <=  0
21) - 120 Y1 + Z15 <=  0
22) - 100 Y2 + Z22 <=  0
23) - 140 Y2 + Z23 <=  0
24) - 200 Y2 + Z24 <=  0
25) - 120 Y2 + Z25 <=  0
26) - 140 Y3 + Z33 <=  0
27) - 200 Y3 + Z34 <=  0
28) - 120 Y3 + Z35 <=  0
29) - 200 Y4 + Z44 <=  0
30) - 120 Y4 + Z45 <=  0
31) - 120 Y5 + Z55 <=  0
32) - X1 + Z11 + Z12 + Z13 + Z14 + Z15 =    0
33) - X2 + Z22 + Z23 + Z24 + Z25 =    0
34) - X3 + Z33 + Z34 + Z35 =    0
35) - X4 + Z44 + Z45 =    0
36) - X5 + Z55 =    0
37) - I1 + W12 + W13 + W14 + W15 =    0
38) - I2 + W23 + W24 + W25 =    0
39) - I3 + W34 + W35 =    0
40) - I4 + W45 =    0

```

The optimal solution value of the linear programming relaxation of model (DLS) for these data is 4814.51610. The optimal solution value of the linear programming relaxation of the multicommodity reformulation of (DLS) is 5090 and is integer. In general, the multicommodity flow reformulation does not give an extended polyhedral representation of the convex hull of solutions to (UNF).

However, for the dynamic lot sizing model, we do get an extended polyhedral representation. See Barany, Van Roy, and Wolsey [35] and Bilde and Krarup [51].

Another reformulation proposed by Rardin and Choe [376] is the arc-path reformulation. Let  $P_k$  for all  $k \in D$  index all of the directed paths from vertex 0 to vertex  $k$ . For each  $l \in P_k$ , let  $w_{lk}$  denote the flow of commodity  $k$  along path  $l$  and let  $\Omega_l$  denote the arcs in path  $l$ .

$$\begin{aligned}
 \min \quad & \sum_{(i,j) \in \mathcal{A}} c_{ij}x_{ij} + \sum_{(i,j) \in \mathcal{A}} f_{ij}y_{ij} \\
 (APUNF) \quad \text{s.t.} \quad & \sum_{l \in P_k} w_{lk} = d_k, \quad k \in D \\
 & \sum_{\{l \in P_k \mid (i,j) \in \Omega_l\}} w_{lk} \leq d_k y_{ij}, \quad (i,j) \in \mathcal{A}, k \in D \\
 & \sum_{k \in D} \sum_{\{l \in P_k \mid (i,j) \in \Omega_l\}} w_{lk} = x_{ij}, \quad (i,j) \in \mathcal{A} \\
 & x, y, w \geq 0 \\
 & y_{ij} \in \{0,1\}, \quad (i,j) \in \mathcal{A}
 \end{aligned}$$

The optimal value of the linear programming relaxation of  $(APUNF)$  is equal to the optimal value of the linear programming relaxation of the  $(MCUNF)$ . This result is a direct consequence of the next proposition.

**Proposition 16.10** *The vector  $(\bar{x}, \bar{y}, \bar{z})$  is a feasible solution of  $(MCUNF)$  if and only if  $(\bar{x}, \bar{y}, \bar{w})$  is a feasible solution to  $(APUNF)$  where*

$$\sum_{\{l \in P_k \mid (i,j) \in \Omega_l\}} \bar{w}_{lk} = \bar{z}_{ijk}.$$

We leave it as Exercise 16.10 to discuss the advantages and disadvantages of each reformulation.

The multicommodity reformulation is a compact model. The number of constraints and variables in the multicommodity reformulation is  $O(|V|^3)$ . The arc-path model is not compact because the number of possible paths in a network is not a polynomial function of the number of arcs and vertices. In the section on branch-and-price we show that the columns in the arc-path reformulation can be generated on-the-fly, as needed.

Multi-commodity flow reformulations have been used to strengthen formulations for the traveling salesman problem. See Claus [95] and Wong [464]. In fact, Swart [417] gave a compact formulation for the traveling salesman problem in auxiliary variables which he claimed had integer vertices. The significance of this claim is that if a compact formulation can be given for the traveling salesman problem with integer vertices, then the traveling salesman problem can be solved by linear programming. The existence of linear programming algorithms with polynomial running time in the size of the problem input then implies the existence of a polynomial time algorithm for the traveling salesman problem. Unfortunately, his claim of integer vertices turned out to be false. Magnanti [306] discusses a number of alternative formulations for vehicle fleet planning, including models based on (*MCUNF*).

### 16.2.2 Leontief Flow Methods

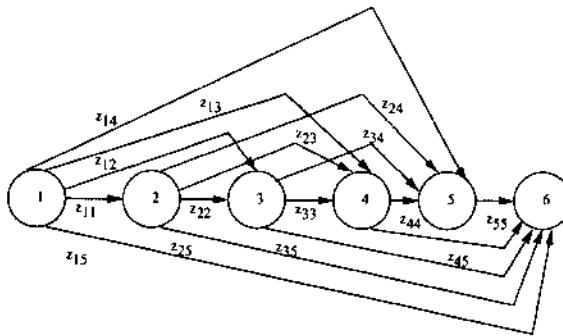
We motivate the reformulation concept in this section using the single product dynamic lot size model (*DLS*) given in equations (16.17)–(16.22). The dynamic lot size model is a specialized example of the uncapacitated network flow problem. In the graph for this model (refer back to Figure 16.2) there are production arcs and inventory arcs. This model has the following important property. If there is an optimal solution to (*DLS*), then there is an optimal  $(\bar{x}, \bar{I})$  such that  $\bar{I}_{t-1}\bar{x}_t = 0$ ,  $t = 1, \dots, T$ . In words, if there is an optimal solution, there is an optimal solution with the property that if there is positive inventory at the beginning of time period  $t$ , then there is no production in time period  $t$ . This famous property is due to Wagner and Whitin [446]. The following proposition is an immediate consequence of this property. Denote the convex hull of solutions to the dynamic lot size model (*DLS*) by  $\text{conv}(\text{DLS})$ . The proof of the proposition is left as Exercise 16.19.

**Proposition 16.11** *If  $(\bar{x}, \bar{I}, \bar{y})$  is an extreme point of  $\text{conv}(\text{DLS})$ , then*

$$\begin{aligned}\bar{x}_t &\in \{0, \tau_{tt}, \tau_{t,t+1}, \dots, \tau_{tT}\}, & t = 1, \dots, T \\ \bar{I}_t &\in \{0, \tau_{t+1,t+1}, \tau_{t+1,t+2}, \dots, \tau_{t+1,T}\}, & t = 1, \dots, T.\end{aligned}$$

In each time period  $t$ , a production decision is made. The decision is to either produce nothing (if there is positive inventory), or to produce a quantity equal to the demand for a consecutive number of time periods beginning with period  $t$ . For example, if there is production in time period  $t$ , then  $x_t = \tau_{tk}$  where  $k \in \{t, \dots, T\}$ . Then demand is covered for periods  $t$  through  $k$  and another

**Figure 16.3** Shortest Route Formulation for Five Period Dynamic Lot Size Model



production decision is not required until period  $k + 1$ . This sequence of production decision can be modeled graphically. Define a new graph with vertex set  $V = \{1, \dots, T+1\}$  and for each vertex  $t = 1, \dots, T$ , define a set of directed arcs  $(t, j), j = t+1, \dots, T+1$ . This graph is illustrated in Figure 16.3 for a five time period model. Assume a supply of one unit at vertex one and a demand of one unit at vertex  $T+1$ . Associated with each arc  $(t, k+1)$  is a variable  $z_{tk}$ . The variable  $z_{tk}$  has the interpretation that when  $z_{tk} = 1$  there is production in period  $t$  to satisfy demand in periods  $t$  through  $k$ . Another production decision is required in period  $k+1$ . Using this variable definition the conservation of flow equations for this network are

$$\sum_{t=1}^T z_{1t} = 1 \quad (16.23)$$

$$-\sum_{k=1}^{t-1} z_{k,t-1} + \sum_{k=t}^T z_{tk} = 0, \quad t = 2, \dots, T \quad (16.24)$$

$$-\sum_{t=1}^T z_{tT} = -1. \quad (16.25)$$

It follows from Proposition 16.11 that there is a one to one correspondence between paths in this network and extreme points of the polytope  $\text{conv}(DLS)$ . To see that this is the case observe the variable correspondence

$$z_{tk} = 1 \leftrightarrow x_t = \tau_{tk}. \quad (16.26)$$

Thus, each path specifies a feasible set of production decisions. For example, the path from vertex one to vertex six using arcs  $(1,3)$  and  $(3,6)$  corresponds

to the production schedule  $x_1 = d_{12}$  and  $x_3 = d_{35}$ . For each period  $t$ , there is a set of variables  $z_{tk}$ ,  $k \in \{t, \dots, T\}$  corresponding to each possible production decision. This gives

$$x_t = \sum_{k=t}^T z_{tk} \tau_{tk}, \quad t = 1, \dots, T \quad (16.27)$$

$$I_t = \sum_{k=1}^t \sum_{j=t+1}^T \tau_{t+1,j} z_{kj}, \quad t = 1, \dots, T. \quad (16.28)$$

The constraints (16.23)-(16.25) are the constraints for a shortest route problem. Refer back to (14.11)-(14.15) for more on the shortest route problem. This is not surprising since the Wagner-Whitin algorithm is a *dynamic programming* algorithm for solving the lot sizing problem. See Nemhauser [351] for background on dynamic programming. The relationship between dynamic programming and shortest route problems is not the relevant issue. See Martin, Rardin, and Campbell [317] for background material on the relationship between dynamic programming and shortest route problems. What is relevant, is that the one to one correspondence between paths in the shortest route problem and extreme points of  $\text{conv}(DLS)$  yields an extended polyhedral representation of the dynamic lot size polytope. Equations (16.27)-(16.28) provide the exact relationship between the original production and inventory variables and the auxiliary variables corresponding to the arcs of the shortest route problem. The reformulated model with auxiliary variables is

$$\min \sum_{t=1}^T (c_t x_t + h_t I_t + f_t y_t) \quad (16.29)$$

$$(RDLS) \quad \text{s.t.} \quad \sum_{t=1}^T z_{1t} = 1 \quad (16.30)$$

$$- \sum_{k=1}^{t-1} z_{k,t-1} + \sum_{k=t}^T z_{tk} = 0, \quad t = 2, \dots, T \quad (16.31)$$

$$\sum_{k=t}^T z_{tk} \leq y_t, \quad t = 1, \dots, T \quad (16.32)$$

$$- \sum_{k=t}^T z_{tk} \tau_{tk} + x_t = 0, \quad t = 1, \dots, T \quad (16.33)$$

$$- \sum_{k=1}^t \sum_{j=t+1}^T \tau_{t+1,j} z_{kj} + I_t = 0, \quad t = 1, \dots, T \quad (16.34)$$

$$x_t, I_t \geq 0, \quad t = 1, \dots, T \quad (16.35)$$

$$y_t \in \{0, 1\}, \quad t = 1, \dots, T. \quad (16.36)$$

Constraint (16.25) is not carried along because it is redundant.

**Proposition 16.12** *Constraint set (16.30)-(16.36) is an extended polyhedral representation of  $\text{conv}(DLS)$ .*

We leave the details of this proof as Exercise 16.23. A more general result is provided later in this section. In formulation (RDLS) notice that the fixed charge constraints (16.32) are written as  $\sum_{k=t}^T z_{tk} \leq y_t$  for  $t = 1, \dots, T$  rather than  $z_{tk} \leq y_t$ ,  $t = 1, \dots, T$ ,  $k = t, \dots, T$ . This is valid because the maximum flow into vertex  $t$  is one unit. Therefore, the flow out of vertex  $t$  which is given by  $\sum_{k=t}^T z_{tk}$  cannot exceed one unit. Writing the fixed charge constraints this way is crucial. If the binary requirements on the  $y_t$  variables are replaced by the condition that  $y_t$  is a nonnegative integer, then the constraint matrix for (RDLS) is pre-Leontief. A matrix is pre-Leontief if and only if there is at most one positive nonzero element per column. Because the right hand sides for these constraints are nonnegative, the inequality system corresponding to these constraints is called a *pre-Leontief substitution system*. The properties of these systems have been investigated by Veinott [445] and Koehler, Whinston, and Wright [268]. See the discussion of Leontief matrices in Chapter 14. If the fixed charge constraints are written as  $z_{tk} \leq y_t$ ,  $t = 1, \dots, T$ ,  $k = t, \dots, T$ , the constraint matrix is no longer pre-Leontief. This is significant in light of Corollary 14.36 which implies that the polyhedron of the linear relaxation of (RDLS) is an integer polyhedron. (It is shown in Martin, Rardin, and Campbell [317] that adding binary constraints on the  $y_t$  does not destroy the integrality of the polyhedron. From an intuitive standpoint, this makes sense since the fixed charge costs are nonnegative, and in any solution it is never optimal to have  $y_{it} > 1$ .) Thus, we can solve (RDLS) as a linear program and do not need to resort to branch and bound.

An example formulation for five product model is given below. We again use the data from Example 12.20 for product one.

```

MIN      150 Y1 + 140 Y2 + 160 Y3 + 160 Y4 + 160 Y5 + 7 X1 + 7 X2
+ 8 X3 + 7 X4 + 7 X5 + I1 + I2 + 2 I3 + 2 I4
SUBJECT TO
 2) Z11 + Z12 + Z13 + Z14 + Z15 =   1
 3) - Z11 + Z22 + Z23 + Z24 + Z25 =   0

```

- 4) - Z12 - Z22 + Z33 + Z34 + Z35 = 0
- 5) - Z13 - Z23 - Z33 + Z44 + Z45 = 0
- 6) - Z14 - Z24 - Z34 - Z44 + Z55 = 0
- 7) - Y1 + Z11 + Z12 + Z13 + Z14 + Z15 <= 0
- 8) - Y2 + Z22 + Z23 + Z24 + Z25 <= 0
- 9) - Y3 + Z33 + Z34 + Z35 <= 0
- 10) - Y4 + Z44 + Z45 <= 0
- 11) - Y5 + Z55 <= 0
- 12) - X1 + 60 Z11 + 160 Z12 + 300 Z13 + 500 Z14 + 620 Z15 = 0
- 13) - X2 + 100 Z22 + 240 Z23 + 440 Z24 + 560 Z25 = 0
- 14) - X3 + 140 Z33 + 340 Z34 + 460 Z35 = 0
- 15) - X4 + 200 Z44 + 320 Z45 = 0
- 16) - X5 + 120 Z55 = 0
- 17) - I1 + 100 Z12 + 240 Z13 + 440 Z14 + 560 Z15 = 0
- 18) - I2 + 140 Z13 + 340 Z14 + 460 Z15 + 140 Z23  
+ 340 Z24 + 460 Z25 = 0
- 19) - I3 + 200 Z14 + 320 Z15 + 200 Z24 + 320 Z25  
+ 200 Z34 + 320 Z35 = 0
- 20) - I4 + 120 Z15 + 120 Z25 + 120 Z35 + 120 Z45 = 0

In the multiproduct, capacitated case, this formulation does not necessarily give integer solutions, but it gives a very tight formulation. The eight product, six period model mentioned in Example 15.4 in the previous chapter requires about four seconds to find and prove optimality on a 200 Mhz Pentium Pro based machine running Windows NT workstation 4.0.

The general reformulation approach we have outlined here applies to any problem where the extreme points of the convex hull of solutions correspond to paths in a shortest route problem and a linear transformation can be constructed from the space of the auxiliary variables to the original variables. Or, more loosely speaking, to any problem that can be solved as a shortest route dynamic program. See Eppen and Martin [139] for other examples of lot sizing problems reformulated using the shortest route approach. Martin [315] gives other problem categories. Gouveia [205] has applied this basic technique to the minimum spanning and Steiner tree problem with hop constraints.

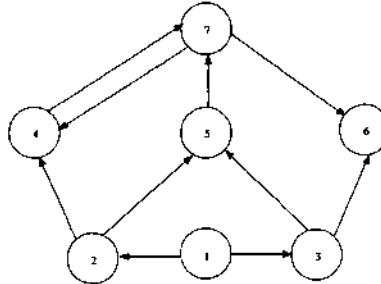
Can this approach be extended to more general uncapacitated network flow problems? Before generalizing, let's view the shortest path reformulation of the dynamic lot size model as a type of multicommodity reformulation. The multicommodity reformulation of the dynamic lot size model developed in the previous subsection is based on creating a distinct commodity for each demand vertex. However, it is certainly possible to extend this idea by *creating a commodity for each possible flow on an arc*. That is, create a "commodity" for

the following flow possibilities

$$\{d_{11}, \dots, d_{1T}, d_{22}, \dots, d_{2T}, \dots, d_{T-1,T-1}, d_{T-1,T}, d_{TT}\}$$

in the dynamic lot size model. Then for a  $T$  period problem there are  $T(T+1)/2$  "commodities." In our notation,  $\tau_{tk}$  is a real number representing the sum of the demands from periods  $t$  through  $k$ . We are a bit loose here and also think of  $\tau_{tk}$  as a unit of the commodity which is the demand for vertices  $t$  through  $k$ . Because of the Wagner-Whitin property, we do not have to consider commodity combinations such as  $d_3 + d_7$ . Initially, at the vertex labeled 0 in Figure 16.2, there is a supply of one unit of commodity  $d_{1T}$ . The production decision is how to "split" this commodity. If the production decision is to produce for periods one through  $t$ , then we enter period  $t+1$  with commodity  $\tau_{t+1,T}$  with another production or split decision to make. Because of the special structure of the dynamic lot size network, it is not necessary to write out any conservation of flow constraints for the commodities. For example, if at vertex 0, the decision is made to produce for periods one through  $t$ , then commodity  $d_{1t}$  is sent to vertex one, commodity  $d_{2t}$  is sent to vertex two, commodity  $d_{3t}$  to vertex three, etc, and finally commodity  $d_t$  is sent to vertex  $t$ . There is no other possibility for periods 1 through  $t$ . That is, a production decision  $\tau_{tk}$  completely determines the flow in and out for vertices  $t+1$  through  $k$ .

There are two issues to address in extending this idea to more general networks. First, in the dynamic lot size model, we used the property that if there is production in a time period there is production for a consecutive number of time periods. This restricted the number of "commodities" considered. In the general case, it may be possible for an arc flow to equal an arbitrary subset of commodity demands. Thus, the commodities are the power set of the set of demand vertices. Second, in the dynamic lot size model, the production decisions uniquely determined the flows of all the commodities. This is not true in general. Consider the graph in Figure 16.4. In Figure 16.4 vertex one is a source vertex with two units, vertex four is a demand vertex of one unit and vertex six is a demand vertex of one unit. The commodity consisting of the demand pair  $\{4, 6\}$  could be sent from vertex one to vertex two. Assume at vertex two the decision is made to split the commodity  $\{4, 6\}$  into two separate commodities  $\{4\}$  and  $\{6\}$ . Unlike the dynamic lot size case, this split decision does not determine uniquely what happens next. For example, commodity  $\{6\}$  could follow the path  $2 \rightarrow 4 \rightarrow 7 \rightarrow 6$  or, it could follow the path  $2 \rightarrow 5 \rightarrow 7 \rightarrow 6$ . Similarly, there are two distinct paths that commodity  $\{4\}$  can follow in going from vertex two to vertex four. In the more general case, both split and routing (or send) decisions must be considered in order to generate an extended polyhedral representation of  $\text{conv}(\Gamma)$  where  $\Gamma = \{(x, y) \mid (x, y) \text{ satisfies (16.13)} - (16.16)\}$ . Now for a more rigorous development of these ideas.

**Figure 16.4** Example Uncapacitated Network Flow Problem

First, it is necessary to characterize the extreme points of  $\text{conv}(\Gamma)$ . Erickson, Monma, and Veinott [142] have shown that if  $(\bar{x}, \bar{y})$  is an extreme point of  $\text{conv}(\Gamma)$  then for every arc  $(i, j)$  there exists an  $I \subseteq D$  such that

$$\bar{x}_{ij} = 0 \quad \text{or}, \quad \bar{x}_{ij} = \sum_{l \in I} d_l. \quad (16.37)$$

That is, the extreme points of  $\text{conv}(\Gamma)$  are characterized by arc flows equal to zero, or the sum of the demands for some subset of demand vertices. In graph terms, the subgraph induced by arcs with positive flows is a forest. Use this result to extend the multicommodity reformulation as follows. Rather than define a commodity for each vertex, define a “commodity” for every subset of demand vertices. Then the number of commodities is equal to the cardinality of the power set of  $V$ , rather than the cardinality of  $V$ . This idea is used by Erickson, Monma, and Veinott in a *send-and-split algorithm*. Given that we are at vertex  $i$  with commodity  $\sum_{l \in I} d_l$ , one possibility is to *send* the commodity on to an adjacent vertex  $j$ , the second possibility is to *split* the commodity into two new commodities corresponding to the proper subsets  $\sum_{l \in I} d_l$  and  $\sum_{l \in I \setminus J} d_l$ , both now located at the same vertex  $i$ . The send-and-split algorithm developed by Erickson, Monma, and Veinott calculates the cost of these decisions in the following manner. Assume that for every vertex  $j \in V$ , and for every subset  $I \subseteq D$  with  $|I| \leq l$ , the minimum cost of sending  $d_i$  units of flow from all vertices  $i \in I$  to vertex  $j$  is known. Then, given  $J \subset D$  with  $|J| = l + 1$  at vertex  $j$  one of two decisions must be made. Either there is a *send decision* to vertex  $j'$  with a cost of  $f_{j,j'} + c_{j,j'} \sum_{i \in J} d_i$ , or a *split decision* is made and set  $J$  is split into  $J'$  and  $J \setminus J'$  where  $|J'| \geq 1$ . If the split decision is made it is then easy to calculate the cost of sending  $d_i$ ,  $i \in J'$  units to vertex  $j$  using arc  $(j, j_1)$  and sending  $d_i$ ,  $i \in J \setminus J'$  units to vertex  $j$  using arc  $(j, j_2)$  since the minimum cost of sending  $d_i$  units of flow from all vertices  $i \in J'$  to vertex  $j_1$  and the minimum cost of sending  $d_i$  units of flow from all vertices  $i \in J \setminus J'$  to vertex

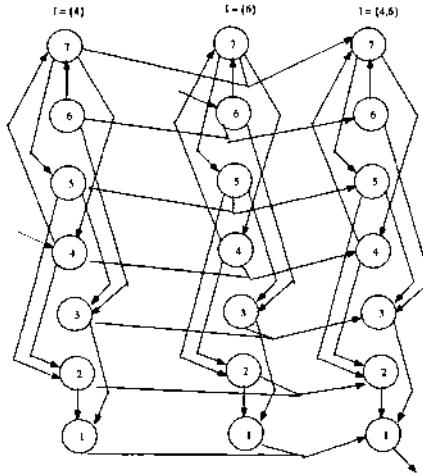
$j_2$  have already been calculated. Proceeding recursively, we can determine the minimum cost of sending  $d_i$  units of flow from all vertices  $i \in I$  to vertex  $j \in V$  for sets  $I$  of increasing cardinality until the minimum cost of sending  $d_i$  units of flow from all vertices  $i \in D$  to vertex 0 is known and the uncapacitated version of the problem is solved. Notice that send-and-split is working backwards, in the sense that the recursion is calculating the cost of sending flow from the demand vertices to the single supply vertex 0.

We now outline the general procedure for constructing a send-and-split *hypergraph*. By hypergraph we mean a directed graph where one or more arcs has a single head but multiple tails. At this point, the reader is advised to go back and review the material on Leontief matrices and hypergraphs in Subsection 14.5.2 in Chapter 14. See also Appendix C. The following construction is due to Jeroslow et al. [243]. This reference also contains more information on hypergraphs and their properties.

1. Create vertex set  $W = V \times \mathcal{P}(D)$  for the hypergraph where  $\mathcal{P}(D)$  is the power set of the demand vertices.
2. Create a *split hyperarc* for each possible split decision. These hyperarcs have head vertex  $(i, I)$  and tail vertices  $\{(i, I'), (i, I \setminus I')\}$ . Each split hyperarc always has two tails and the tail weights are always +1. The flow on the split hyperarc is  $z[\{(i, I'), (i, I \setminus I')\}, (i, I)]$ .
3. Create a *send hyperarc* for each possible send decision. For each  $(i, j) \in A$  and  $I \subseteq D$  the send hyperarc has head  $(i, I)$  and tail  $(j, I)$ . Note the hyperarc points in the reverse direction of the network arc  $(i, j)$ . Tail weights are +1. Denote the flow on the send hyperarc by  $z[(j, I), (i, I)]$ .
4. Create a *source hyperarc* directed into every vertex  $(i, \{i\})$  all  $i \in D$ . These hyperarcs are also assigned a zero cost in the objective function. Denote the flow on the source hyperarcs by  $z[\emptyset, (i, \{i\})]$ .
5. Create a unit demand at vertex  $(0, D)$ .

The send-and-split hypergraph for the graph in Figure 16.4 is given in Figure 16.5. The flows on the hyperarcs in the send-and-split hypergraph determine the value of the flows in the graph corresponding to problem (UNF) through the equation

$$x_{ij} = \sum_{I \subseteq D} \left[ \sum_{h \in I} d_h \right] z[(j, I), (i, I)] \quad (16.38)$$

**Figure 16.5** Send-and-Split Hypergraph for Network of Figure 16.4

Combining (16.38) with the hypergraph flow balance constraints gives the reformulated or extended polyhedral representation (*RUNF*). The auxiliary  $z$  variables are the flow variables in the send-and-split hypergraph.

$$\min \sum_{(i,j) \in \mathcal{A}} (c_{ij}x_{ij} + f_{ij}y_{ij}) \quad (16.39)$$

$$(RUNF) \quad \text{s.t.} \quad \begin{aligned} & \sum_{(i,j) \in \mathcal{A}} z[(j, I), (i, I)] + \sum_{I' \subset I} z[\{(i, I'), (i, I \setminus I')\}, (i, I)] \\ & - \sum_{(j,i) \in \mathcal{A}} z[(i, I), (j, I)] - \sum_{I \subset H} z[\{(i, I), (i, H \setminus I)\}, (i, H)] \\ & = \begin{cases} 1, & (i, I) = (0, D) \\ -z[\emptyset, (i, I)], & (i, I) = (i, \{i\}) \\ 0, & \text{all other } (i, I) \end{cases} \end{aligned} \quad (16.40)$$

$$- \sum_{I \subseteq D} z[(j, I), (i, I)] + y_{ij} \geq 0, \quad (i, j) \in \mathcal{A} \quad (16.41)$$

$$- \sum_{I \subseteq D} \left( \sum_{h \in I} d_h \right) z[(j, I), (i, I)] + x_{ij} = 0, \quad (i, j) \in \mathcal{A} \quad (16.42)$$

$$y, z \geq 0 \quad (16.43)$$

Problem (*RUNF*) has considerable structure. The matrix consisting of constraints (16.40)-(16.43) is a *pre-Leontief* matrix. Again, refer to the develop-

ment in Subsection 14.5.2 in Chapter 14. In general, flow problems on hypergraphs such as the send-and-split hypergraph have pre-Leontief constraint matrices and Jeroslow et al. call these *Leontief substitution flow problems*. Given integer demands  $d_i, i \in D$ , it follows from Corollary 14.36 that the polyhedron corresponding to the pre-Leontief substitution system defined by (16.40)-(16.43) is an integer polyhedron. It is possible to strengthen this result and show that requiring the  $z$  to be binary does not destroy the integrality of the polyhedron. Lemma 16.13 is from Jeroslow et al. [243]

**Lemma 16.13** *If  $(\bar{x}, \bar{y}, \bar{z})$  is an extreme point of the polyhedron corresponding to the pre-Leontief substitution system defined by (16.40)-(16.43), then  $\bar{z}$  is binary.*

An immediate consequence of Lemma 16.13 and constraint (16.41) is that if  $(\bar{x}, \bar{y}, \bar{z})$  is an extreme point of the polyhedron corresponding to the pre-Leontief substitution system defined by (16.40)-(16.43), then  $\bar{y}$  is integer.

**Proposition 16.14** *The pre-Leontief substitution system defined by (16.40)-(16.43) is an extended polyhedral representation of  $\text{conv}(\Gamma)$  where  $\Gamma = \{(x, y) | (x, y) \text{ satisfies (16.13) - (16.15), } y \text{ integer}\}$ .*

**Proof:** Apply Proposition 16.7. First show that if  $(\bar{x}, \bar{y}, \bar{z})$  is an element of the polyhedron defined by (16.40)- (16.43) then  $(\bar{x}, \bar{y}) \in \text{conv}(\Gamma)$ . Assume without loss, that  $(\bar{x}, \bar{y}, \bar{z})$  is also an extreme point of the polyhedron. To see that  $(\bar{x}, \bar{y})$  is in  $\text{conv}(\Gamma)$  create an aggregate constraint for each  $l \in V$ , by summing all the constraints in (16.40) for  $i = l$  using multipliers of  $-(\sum_{h \in I} d_h)$ . With this weighting all split variables cancel in the aggregate constraint leaving

$$\begin{aligned} \sum_{(j,i) \in A} \sum_{I \subseteq D} \left( \sum_{h \in I} d_h \right) \bar{z}[(l, I), (j, I)] &= \\ \sum_{(l,j) \in A} \sum_{I \subseteq D} \left( \sum_{h \in I} d_h \right) \bar{z}[(j, I), (l, I)] &= \begin{cases} -\sum_{h \in D} d_h, & l = 0 \\ d_l \bar{z}[\emptyset, (l, \{l\})], & l \in D \\ 0, & l \in T. \end{cases} \end{aligned} \quad (16.44)$$

This aggregation process is illustrated in Example 16.15. To see that the right hand sides of (16.44) are correct, observe that the right hand sides of (16.40) are all 0 except when  $l = 0$ , or  $l \in D$ . The first of these produces  $-\sum_{h \in D} d_h$  because of the unit demand at vertex  $(0, D)$ . For  $l \in D$ , the right hand side in the aggregate constraint is  $d_l \bar{z}[\emptyset, (l, \{l\})]$ . By Lemma 16.13, in every extreme

point solution of the polyhedron defined by (16.40)-(16.43),  $\bar{z}[\emptyset, (l, \{l\})] = 1.0$ . Therefore  $d_l \bar{z}[\emptyset, (l, \{l\})] = d_l$ . Then, using the definition of  $x_{ij}$  in (16.42) and recalling that send hyperflows  $x$  are reverse to  $z$  flows, we see that (16.44) implies (16.13). Constraint (16.42) and (16.41) imply that (16.14) is satisfied. Therefore,  $(\bar{x}, \bar{y})$  is a feasible solution to (16.13)-(16.15). Since  $\bar{y}$  is also integer by Lemma 16.13,  $(\bar{x}, \bar{y}) \in \Gamma \subseteq \text{conv}(\Gamma)$ .

Now verify part 2 of Proposition 16.7 and show that for every generator  $(\bar{x}, \bar{y})$  of  $\text{conv}(\Gamma)$  there is a corresponding  $\bar{z}$  such that  $(\bar{x}, \bar{y}, \bar{z})$  is in the feasible region of  $(RUNF(x, y, z))$ . If  $(\bar{x}, \bar{y})$  is an extreme point solution in  $\text{conv}(\Gamma)$  then the graph induced by the positive values of  $\bar{x}$  is a forest. Clearly this implies a corresponding combination of send-and-split decisions which correspond to an extreme (binary) solution in the send-and-split hypergraph.

Finally, each extreme direction of  $\text{conv}(\Gamma)$  corresponds to a directed cycle in  $G = (V, A)$ . For all  $I \subseteq D$ , each of these directed cycles also exist in the hypergraph (in reverse direction). Thus, there is a set of send decisions in the hypergraph for each extreme ray of  $\text{conv}(\Gamma)$ .  $\square$

In this proposition we have assumed that  $y$  is integer in  $\text{conv}(\Gamma)$  rather than binary. See constraint (16.16) in model  $(UNF)$ . This is because it is not necessarily the case that  $y$  is binary in extreme point solutions of (16.40)-(16.43). This does not cause a problem in practice since in any real application the fixed costs on the arcs are nonnegative so the binary condition in (16.16) can be replaced by simply requiring  $y$  to be an integer vector.

**Example 16.15** Consider vertex  $l = 2$  in Figure 16.5. The constraint for vertex two when  $I = \{4, 6\}$  is

$$\begin{aligned} & z[(4, \{4, 6\}), (2, \{4, 6\})] + z[(5, \{4, 6\}), (2, \{4, 6\})] \\ & + z[\{(2, \{4\}), (2, \{6\})\}, (2, \{4, 6\})] - z[(2, \{4, 6\}), (1, \{4, 6\})] = 0. \end{aligned} \quad (16.45)$$

The constraint for vertex two when  $I = \{4\}$  is

$$\begin{aligned} & z[(4, \{4\}), (2, \{4\})] + z[(5, \{4\}), (2, \{4\})] \\ & - z[\{(2, \{4\}), (2, \{6\})\}, (2, \{4, 6\})] - z[(2, \{4\}), (1, \{4\})] = 0. \end{aligned} \quad (16.46)$$

The constraint for vertex two when  $I = \{6\}$  is

$$\begin{aligned} & z[(4, \{6\}), (2, \{6\})] + z[(5, \{6\}), (2, \{6\})] \\ & - z[\{(2, \{4\}), (2, \{6\})\}, (2, \{4, 6\})] - z[(2, \{6\}), (1, \{6\})] = 0. \end{aligned} \quad (16.47)$$

Aggregating these constraints with a multiplier of  $-(d_4 + d_6)$  on (16.45), a multiplier of  $-d_4$  on (16.46) and  $-d_6$  on (16.47) gives

$$\begin{aligned} & -(d_4 + d_6)z[(4, \{4, 6\}), (2, \{4, 6\})] - d_4z[(4, \{4\}), (2, \{4\})] \\ & - d_6z[(4, \{6\}), (2, \{6\})] - (d_4 + d_6)z[(5, \{4, 6\}), (2, \{4, 6\})] \\ & \quad - d_4z[(5, \{4\}), (2, \{4\})] - d_6z[(5, \{6\}), (2, \{6\})] \\ & +(d_4 + d_6)z[(2, \{4, 6\}), (1, \{4, 6\})] + d_4z[(2, \{4\}), (1, \{4\})] \\ & \quad + d_6z[(2, \{6\}), (1, \{6\})] = 0. \end{aligned} \quad (16.48)$$

Using the variable definition in (16.38), the aggregate constraint (16.48) implies  $-x_{24} - x_{25} + x_{12} = 0$  which is the conservation of flow equation for vertex two in Figure 16.4.

Leontief substitution flow models can be used to generate auxiliary variables in order to tighten a number of different integer programming models. Examples include various types of lot sizing models such as those with multiple facilities, backlogging and changeover costs, optimization problems on recursively defined graphs, and network flow problems on planar graphs. For such examples see Eppen and Martin [139] and Jeroslow et al. [243] [243]. In general, formulation (RUNF) is not polynomial in the number of demand vertices because all subsets of  $D$  have to be considered. However, Erickson, Monma, and Veinott show that when the graph of (UNF) can be embedded in the plane with the source and demand vertices on at most  $\beta$  faces, then only  $O(\beta|D|^2)$  demand subsets must be considered and (RUNF) is a compact extended formulation for fixed  $\beta$ . Many of the application problems listed above have this property. The Leontief flow reformulations of this section generalize the earlier work of Martin [315] and Martin, Rardin and Campbell [317].

### 16.2.3 Disjunctive Methods

The ideas in this section are due to Balas [20, 22, 23] and Jeroslow and Lowe [249, 250]. Define a set of polyhedra by  $\Gamma_i = \{x \in \mathbb{R}^n \mid A_i x \geq b^i\}$ . Earlier, we saw that one way to model the union of the polyhedra  $\Gamma_i$  for  $i = 1, \dots, m$  as a mixed integer program was

$$A_i x \geq b^i - (1 - y_i)M e, \quad i = 1, \dots, m \quad (16.49)$$

$$\sum_{i=1}^m y_i = 1 \quad (16.50)$$

$$y_i \in \{0, 1\}, \quad i = 1, \dots, m \quad (16.51)$$

where  $M$  is the *big M* used in integer programming modeling and  $e$  is a column vector of ones with dimension equal to the dimension of  $b^i$ . The *disjunctive formulation* for this problem creates a set of auxiliary variables  $z^i$ , for each polyhedron. The disjunctive formulation using the auxiliary variables is

$$A_i z^i \geq b^i z_{i0}, \quad i = 1, \dots, m \quad (16.52)$$

$$\sum_{i=1}^m z^i = x \quad (16.53)$$

$$\sum_{i=1}^m z_{i0} = 1 \quad (16.54)$$

$$z_{i0} \geq 0, \quad i = 1, \dots, m \quad (16.55)$$

$$z_{i0} \in \{0, 1\}, \quad i = 1, \dots, m. \quad (16.56)$$

As long as the polyhedra are not empty, the polyhedron defined by constraints (16.52)-(16.55) is an extended polyhedral representation of  $\text{conv}(\cup_{i=1}^m \Gamma_i)$ . This important proposition is due to Balas [20, 23].

**Proposition 16.16** *If  $\Gamma_i = \{x \in \mathbb{R}^n \mid A_i x \geq b^i\} \neq \emptyset$  for  $i = 1, \dots, m$ , then (16.52)-(16.55) is an extended polyhedral representation of  $\text{conv}(\cup_{i=1}^m \Gamma_i)$ .*

The linear programming relaxation, (16.52)-(16.55), of the disjunctive formulation defines an integer polyhedron.

**Proposition 16.17** *If  $(\bar{z}^0, \bar{z}^1, \dots, \bar{z}^m)$  is an extreme point of the polyhedron (16.52)-(16.55), then  $\bar{z}^0$  is a binary vector.*

The result of Balas given in Proposition 16.16 can be used to generate stronger formulations of *(MIP)*. Assume for example, that in formulation *(MIP)* the integer variables indexed by  $I$  are binary variables. If  $k \in I$  then in every feasible solution to *(MIP)*, either  $x_k = 0$  or  $x_k = 1$ . Another way to state this is to define

$$\Gamma_1 = \{x \mid Ax \geq b, Bx \geq d, x \geq 0, x_j \leq 1, j \in I, x_k = 1\}$$

$$\Gamma_2 = \{x \mid Ax \geq b, Bx \geq d, x \geq 0, x_j \leq 1, j \in I, x_k = 0\}.$$

and observe that

$$\text{conv}((MIP)) \subseteq \Gamma_1 \cup \Gamma_2 \subseteq \text{conv}(\Gamma_1 \cup \Gamma_2) \subseteq \text{conv}(\overline{(MIP)}).$$

By Proposition 16.16 an extended polyhedral representation of  $\text{conv}(\Gamma_1 \cup \Gamma_2)$  with auxiliary variables is

$$\begin{aligned}
 & \min c^\top x \\
 \text{s.t. } & Az^1 \geq bz_{10} \\
 & Az^2 \geq bz_{20} \\
 & Bz^1 \geq dz_{10} \\
 & Bz^2 \geq dz_{20} \\
 & z^1 + z^2 - x = 0 \\
 & z_{1j} \leq z_{10}, \quad j \in I \\
 & z_{2j} \leq z_{20}, \quad j \in I \\
 & z_{1k} - z_{10} = 0 \\
 & z_{2k} = 0 \\
 & z_{10} + z_{20} = 1 \\
 & z^1, z^2, z_{10}, z_{20} \geq 0
 \end{aligned}$$

Using  $z^1 + z^2 - x = 0$  to substitute out the  $x$  variables in the objective function gives a formulation with approximately twice the number of variables and constraints as the original formulation. Repeating this process for each binary variable leads to a formulation that is tight. We illustrate this idea for the generalized assignment problem.

**Example 16.18** Consider the generalized assignment problem from Example 15.7 in Chapter 15. The test problem is

```

MIN    2 X11 + 11 X12 + 7 X21 + 7 X22 + 20 X31 + 2 X32 + 5 X41 + 5 X42
SUBJECT TO
2)  X11 + X12 = 1
3)  X21 + X22 = 1
4)  X31 + X32 = 1
5)  X41 + X42 = 1
6)  3 X11 + 6 X21 + 5 X31 + 7 X41 <= 13
7)  2 X12 + 4 X22 + 10 X32 + 4 X42 <= 10
  
```

The solution of the linear programming relaxation is

OBJECTIVE FUNCTION VALUE
--------------------------

1) 20.32000

VARIABLE	VALUE	REDUCED COST
X11	1.000000	.000000
X21	1.000000	.000000
X31	.240000	.000000
X32	.760000	.000000
X41	.400000	.000000
X42	.600000	.000000

Use the fractional variable  $x_{41}$  for the disjunctive reformulation. The new formulation and solution are below. The auxiliary variables  $z_{1ij}$  are used to define the polyhedron corresponding to  $x_{41} = 1$  and the variables  $z_{2ij}$  are used to define the polyhedron corresponding to  $x_{41} = 0$ . We do not include any constraints of the form  $z_{kij} \leq z_{k0}$  since the assignment constraints  $\sum_{i=1}^2 x_{ij} = 1$  make the simple upper bound constraints on the  $x_{ij}$  variables redundant.

```

MIN      2 X11 + 11 X12 + 7 X21 + 7 X22 + 20 X31 + 2 X32 + 5 X41 + 5 X42
SUBJECT TO
    2) Z111 + Z112 - Z10 = 0
    3) - Z10 + Z121 + Z122 = 0
    4) - Z10 + Z131 + Z132 = 0
    5) - Z10 + Z141 + Z142 = 0
    6) 3 Z111 - 13 Z10 + 6 Z121 + 5 Z131 + 7 Z141 <= 0
    7) 2 Z112 - 10 Z10 + 4 Z122 + 10 Z132 + 4 Z142 <= 0
    8) Z211 + Z221 - Z20 = 0
    9) - Z20 + Z221 + Z222 = 0
   10) - Z20 + Z231 + Z232 = 0
   11) - Z20 + Z241 + Z242 = 0
   12) 3 Z211 - 13 Z20 + 6 Z221 + 5 Z231 + 7 Z241 <= 0
   13) 2 Z212 - 10 Z20 + 4 Z222 + 10 Z232 + 4 Z242 <= 0
   14) Z10 + Z20 = 1
   15) - Z10 + Z141 = 0
   16) Z241 = 0
   17) - X11 + Z111 + Z211 = 0
   18) - X21 + Z121 + Z221 = 0
   19) - X31 + Z131 + Z231 = 0
   20) - X41 + Z141 + Z241 = 0
   21) - X12 + Z112 + Z212 = 0
   22) - X22 + Z122 + Z222 = 0
   23) - X32 + Z132 + Z232 = 0
   24) - X42 + Z142 + Z242 = 0
END

```

## OBJECTIVE FUNCTION VALUE

1) 21.40000

VARIABLE	VALUE	REDUCED COST
X11	1.000000	.000000
X21	.250000	.000000
X22	.750000	.000000
X31	.300000	.000000
X32	.700000	.000000
X41	1.000000	.000000
Z111	1.000000	.000000
Z10	1.000000	.000000
Z121	.250000	.000000
Z122	.750000	.000000
Z131	.300000	.000000
Z132	.700000	.000000
Z141	1.000000	.000000

By using the disjunctive formulation the linear relaxation improves from 20.32 to 21.4. It is left as Exercise 16.9 to show that a disjunctive formulation based on variable  $x_{32}$  yields the optimal integer solution.

As long as the  $\Gamma_i$  are not null, the disjunctive method gives an extended polyhedral representation of  $\text{conv}(\cup_{i=1}^m \Gamma_i)$ . Curiously enough, it is not always the case that either (16.52)-(16.56) or (16.49)-(16.51) are valid modelings of  $\cup_{i=1}^m \Gamma_i$ . Consider the following example.

**Example 16.19**

$$\begin{aligned}\Gamma_1 &= \{(x_1, x_2) \mid x_1 \geq 1, -x_1 \geq -1, x_2 \geq 0\} \\ \Gamma_2 &= \{(x_1, x_2) \mid x_2 \geq 1, -x_2 \geq -1, x_1 \geq 0\}\end{aligned}$$

The disjunctive formulation is

$$\begin{aligned}z_{11} &\geq z_{10} \\ -z_{11} &\geq -z_{10} \\ z_{12} &\geq 0 \\ z_{22} &\geq z_{20} \\ -z_{22} &\geq -z_{20}\end{aligned}$$

$$\begin{aligned}
 z_{21} &\geq 0 \\
 z_{10} + z_{20} &= 1 \\
 z_{11} + z_{21} - x_1 &= 0 \\
 z_{12} + z_{22} - x_2 &= 0 \\
 z_{11}, z_{12}, z_{21}, z_{22} &\geq 0
 \end{aligned}$$

A feasible solution is

$$z_{10} = 1, z_{11} = 1, z_{12} = 1000, z_{21} = 1000, x_1 = 1001, x_2 = 1000$$

with all other variables set to zero. The solution  $(1001, 1000)$  is not in  $\Gamma_1 \cup \Gamma_2$ . If the loose formulation (16.49)-(16.51) is used there is no value of big  $M$  that will work in the constraints  $A_i x \geq b^i - (1 - y_i)eM$  when the variable  $y_i = 0$ . Why?

Jeroslow and Lowe [249] give both necessary and sufficient conditions for modeling  $\cup_{i=1}^m \Gamma_i$  as a mixed integer linear program. A simple sufficient condition for either (16.52)-(16.56) or (16.49)-(16.51) to be MIP representations of  $\cup_{i=1}^m \Gamma_i$  is for each of the  $\Gamma_i$  to be bounded.

### 16.2.4 Separation Based Methods

In Chapter 15 cutting plane algorithms were used to tighten integer programming formulations. Most cutting planes algorithms employ a separation problem. The separation problem is used to determine if the current solution satisfies all of the constraints not explicitly included in the formulation; and if not, to find at least one violated constraint. Often this separation problem is formulated as a linear program. In this subsection, we show that when the separation problem is a linear program it is possible to generate a reformulation with auxiliary variables and the linear relaxation of the reformulated model is equal to the linear relaxation of the formulation with all of the cuts added from separation.

Consider the linear program  $\min \{c^\top x \mid Ax \geq b, x \in \Gamma \subseteq \mathbb{R}^n\}$  where

$$\Gamma = \{x \mid (\alpha^i)^\top x \leq \alpha_0^i, i = 1, \dots, q, x \geq 0\}.$$

In the current setting, the original mixed integer program is

$$\min \{c^\top x \mid Ax \geq b, x \geq 0, x_j \in \mathcal{Z}, j \in I\}$$

and the linear program  $\min \{c^\top x \mid Ax \geq b, x \in \Gamma \subseteq \Re^n\}$  is its linear relaxation with a set of cutting planes  $(\alpha^i)^\top x \leq \alpha_0^i, i = 1, \dots, q$  added to tighten the formulation. In many interesting cases the number of cutting planes in  $\Gamma$  is too large to add all of them explicitly and they are added on-the-fly using a separation algorithm. Often this separation problem can be formulated as a linear program. The linear program

$$(SP(x)) \quad \max \{x^\top \alpha + h^\top \theta \mid F\alpha + H\theta \leq 0, \theta \geq 0\}$$

is a *valid separation linear program* for  $\Gamma$  if, given  $\bar{x} \geq 0$  it has an optimal solution if and only if  $(\alpha^i)^\top \bar{x} \leq \alpha_0^i, i = 1, \dots, q$ . If there exists an optimal separation linear program for  $(SP(x))$  then it is possible to generate an extended polyhedral representation of  $\{x \mid Ax \geq b, x \in \Gamma\}$ . Consider the linear program

$$(RM) \quad \begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq d \\ & x = F^\top z \\ & H^\top z \geq h \\ & x, z \geq 0 \end{aligned}$$

**Proposition 16.20** *Let  $(SP(x))$  be a valid separation linear program for  $\Gamma$ . Then  $\bar{x} \in \{x \mid Ax \geq b, x \in \Gamma\}$  if and only if there exists a  $\bar{z}$  such that  $(\bar{x}, \bar{z})$  is feasible to  $(RM)$ .*

**Proof:** Consider an  $\bar{x}$  such that  $A\bar{x} \geq b$  and  $\bar{x} \geq 0$ . By hypothesis  $(SP(x))$  is a valid separation linear program for  $\Gamma$ . Because  $(SP(x))$  is a linear program over a cone it has an optimal solution if and only if its optimal solution value is zero. This implies that  $(SP(x))$  has an optimal solution if and only if its dual is feasible. The dual is feasible if and only if there exists  $\bar{z} \geq 0$  such that  $x = F^\top \bar{z}, H^\top \bar{z} \geq h$ , i.e.  $(\bar{x}, \bar{z})$  is feasible to  $(RM)$ .  $\square$

**Example 16.21 (Minimum Spanning Tree)** *Let  $G = (V, E)$  be an undirected graph with vertex set  $V = \{1, \dots, n\}$  and edge set  $E$ . For  $S \subseteq V$  let  $\gamma(S) \subset E$  denote all edges of  $E$  with both endpoints in  $S$  and  $\delta(S) \subseteq E$  the set of edges with exactly one endpoint in  $S$ . For  $E' \subset E$  define  $x(E') := \sum_{e \in E'} x_e$ . A spanning tree of  $G$  is a subgraph  $G' = (V, E')$  which is a tree. If there is a weight  $c_e$ , assigned to each edge  $e \in E$  the minimum spanning tree is the spanning tree of minimum weight. A linear programming formulation (MST) for the minimum spanning tree problem on  $G$  is  $\min \{\sum_{e \in E} c_e x_e \mid x \in \Gamma(MST), \sum_{e \in E} x_e = n - 1\}$  where*

$$\Gamma(MST) = \{x \mid x(\gamma(S)) \leq |S| - 1, S \subseteq V, 2 \leq |S| \leq n - 1, x_e \geq 0, e \in E\}.$$

The polyhedron for the minimum spanning tree problem is an integer polyhedron and the optimal linear programming solution to the linear program (MST) gives a minimum weight spanning tree. Cunningham [106], Padberg and Wolsey [367], and Picard and Queyranne [370] show that for each  $k \in V$  it is possible to test if  $0 \leq \bar{x} \leq 1$  violates a constraint in  $\Gamma(MST)$  for some  $S \subseteq V$  with  $k \in S$  by solving a maximum flow problem. A linear program corresponding to the maximum flow problem (see, for example, Rhys [381]) is

$$\max \sum_{e \in E} \bar{x}_e \alpha_e^k - \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i^k \quad (16.57)$$

$$(SP_k(\bar{x})) \quad \text{s.t.} \quad \alpha_e^k \leq \theta_i^k, \quad e \in \delta(i), \quad i \in V \quad (16.58)$$

$$\theta^k \geq 0. \quad (16.59)$$

Given nonnegative  $\bar{x}$ , since the objective function is to be maximized, we need only consider solutions with  $\alpha_{(i,j)}^k = \min\{\theta_i^k, \theta_j^k\}$ . It is fairly simple to show that the extreme rays of the cone defined by (16.58)-(16.59) have 0-1 components. Then for  $0 \leq \bar{x} \leq 1$ ,  $(SP_k(\bar{x}))$  has an optimal solution if and only if  $\bar{x}(\gamma(S)) \leq |S| - 1$ ,  $k \in S$ . Therefore,  $(SP_k(\bar{x}))$  is a valid separation linear program for all cuts which contain vertex  $k$ . The reformulated minimum spanning tree problem with a set of auxiliary variables for each separation problem for each vertex follows.

$$(RMST) \quad \begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & z_{kij} + z_{kji} = x_e, \quad k = 1, \dots, n, \quad e \in \gamma(\{i, j\}) \end{aligned} \quad (16.60)$$

$$\sum_{s>i} z_{kis} + \sum_{h<i} z_{kih} \leq 1, \quad k = 1, \dots, n, \quad i \neq k \quad (16.61)$$

$$\sum_{s>k} z_{kks} + \sum_{h<k} z_{khh} \leq 0, \quad k = 1, \dots, n \quad (16.62)$$

$$x, z \geq 0 \quad (16.63)$$

The auxiliary variables  $z_{kij}$  have a nice interpretation in terms of the graph  $G$ . The variables  $x_e$  correspond to undirected edges in  $G$  while the  $z_{kij}$  correspond to directed edges, i.e. arcs. The variable  $z_{ijk}$  or  $z_{kji}$  will equal one, for some  $k$ , if edge  $x_e$  is in the solution. If this solution has a cycle of undirected edges which contain vertex  $k$ , then by (16.60) there is a cycle of arcs containing vertex  $k$ . However, there cannot be a cycle of arcs which contain vertex  $k$ . If such a cycle exists, and is directed, then the cycle of arcs which contains vertex  $k$  has at least one arc directed out of vertex  $k$ . This is impossible by (16.62) and the

cycle cannot be directed. If the cycle is not directed, then there is at least one vertex  $i$  with two arcs directed out of it. This is impossible by (16.61). Then there cannot be any cycles in any binary solution of (RMST).

The number of variables in the polyhedron  $\Gamma(MST)$  is  $O(n^2)$  with  $O(2^n)$  constraints. The extended formulation of  $\Gamma(MST)$  has  $O(n^3)$  variables and  $O(n^3)$  constraints. An exponential number of constraints were replaced by a polynomial number of variables. See Gouveia [203] for a discussion of formulations for the capacitated minimum spanning tree problem.

**Example 16.22 (Dynamic Lot Sizing Continued)** In Chapter 15 a characterization due to Barany, Van Roy, and Wolsey [34, 35] of the convex hull of integer solutions was given. The characterization was based on the  $(l, S)$  inequalities. Let

$$\begin{aligned} \Gamma(DLS) = \{(x, I, y) \mid & \sum_{t \in S} x_t \leq \sum_{t \in S} \tau_{tl} y_t + I_l, x_t, I_l, y_t \geq 0, \\ & y_t \leq 1, t = 1, \dots, l, S \subseteq \{1, \dots, l\} l = 1, \dots, T\}. \end{aligned}$$

Given  $(\bar{x}, \bar{I}, \bar{y})$ , a simple separation algorithm to discover a violated cut for a given  $l$  is to calculate  $S = \{t \mid \bar{x}_t > \tau_{tl} \bar{y}_t, t = 1, \dots, l\}$ . If  $\sum_{t \in S} (\bar{x}_t - \tau_{tl} \bar{y}_t) \geq \bar{I}_l$  then add the cut. It is easy to see that the set  $S$  is also found through linear programming. A valid separation linear program for each  $l = 1, \dots, T$  is

$$\begin{aligned} \max & \sum_{t=1}^l \alpha_t^1 (\bar{x}_t - \tau_{tl} \bar{y}_{tl}) - \theta_l \bar{I}_l \\ (DLS(l, S)) \quad \text{s.t.} & \alpha_{tl}^1 + \alpha_{tl}^2 \leq \theta_l, t = 1, \dots, l \\ & \alpha_{tl}^1, \alpha_{tl}^2, \theta_l \geq 0, t = 1, \dots, l. \end{aligned}$$

Clearly the optimal solution value of  $(DLS(l, S))$  is 0 for a given  $(\bar{x}, \bar{I}, \bar{y})$  if and only if this solution does not violate any  $(l, S)$  inequalities. Applying Proposition 16.20 to the linear program  $(DLS(l, S))$  for  $l = 1, \dots, T$  gives an extended formulation of  $\Gamma(DLS)$  with auxiliary variables. The auxiliary variables  $z_{tl}$  correspond to the dual variables on the constraints  $\alpha_{tl}^1 + \alpha_{tl}^2 = \theta_l$ .

$$\begin{aligned} z_{tl} & \geq x_t - \tau_{tl} y_t, \quad t = 1, \dots, l, l = 1, \dots, T \\ \sum_{t=1}^l z_{tl} & \leq I_l, \quad l = 1, \dots, T \\ x_t, y_t, I_l & \geq 0, \quad t = 1, \dots, T \\ y_t & \leq 1, \quad t = 1, \dots, T \\ z_{tl} & \geq 0, \quad t = 1, \dots, l, l = 1, \dots, T \end{aligned}$$

In this formulation the  $z_{tl}$  auxiliary variable can be interpreted as the amount of production in period  $t$  used to satisfy demand for periods  $l+1$  or greater. Below is the complete formulation using the separation algorithm auxiliary variables for the five period example from Example 16.9.

```

MIN      150 Y1 + 140 Y2 + 160 Y3 + 160 Y4 + 160 Y5 + 7 X1 + 7 X2 + 8 X3
        + 7 X4 + 7 X5 + I1 + I2 + 2 I3 + 2 I4
SUBJECT TO
  2)  X1 - I1 =    60
  3)  X2 + I1 - I2 =   100
  4)  X3 + I2 - I3 =   140
  5)  X4 + I3 - I4 =   200
  6)  X5 + I4 =   120
  7) - 620 Y1 + X1 <=  0
  8) - 560 Y2 + X2 <=  0
  9) - 460 Y3 + X3 <=  0
 10) - 320 Y4 + X4 <=  0
 11) - 120 Y5 + X5 <=  0
 12)  60 Y1 - X1 + Z11 >=  0
 13) 160 Y1 - X1 + Z12 >=  0
 14) 100 Y2 - X2 + Z22 >=  0
 15) 300 Y1 - X1 + Z13 >=  0
 16) 240 Y2 - X2 + Z23 >=  0
 17) 140 Y3 - X3 + Z33 >=  0
 18) 500 Y1 - X1 + Z14 >=  0
 19) 440 Y2 - X2 + Z24 >=  0
 20) 340 Y3 - X3 + Z34 >=  0
 21) 200 Y4 - X4 + Z44 >=  0
 22) 620 Y1 - X1 + Z15 >=  0
 23) 560 Y2 - X2 + Z25 >=  0
 24) 460 Y3 - X3 + Z35 >=  0
 25) 320 Y4 - X4 + Z45 >=  0
 26) 120 Y5 - X5 + Z55 >=  0
 27) - I1 + Z11 <=  0
 28) - I2 + Z12 + Z22 <=  0
 29) - I3 + Z13 + Z23 + Z33 <=  0
 30) - I4 + Z14 + Z24 + Z34 + Z44 <=  0
 31) Z15 + Z25 + Z35 + Z45 + Z55 <=  0

```

Constraints 2)-11) are the constraints for the loose model (DLS). Constraints 12)-31) represent the extended polyhedral representation of the convex hull of solutions defined by the  $(l, S)$  inequalities. The optimal linear programming relaxation of this formulation is integer since adding the  $(l, S)$  inequalities to model (DLS) gives a tight reformulation.

**Table 16.1** Comparison of Dynamic Lot Sizing Formulations

Model	Number of Variables	Number of Constraints
Model with $(l, S)$ inequalities	$O(T)$	$O(2^T)$
Separation Reformulation	$O(T^2)$	$O(T^2)$
Shortest Route Reformulation	$O(T^2)$	$O(T)$
Multi-commodity Reformulation	$O(T^2)$	$O(T^2)$

Sabinidis and Grossmann [390] use the separation algorithm based reformulation technique to solve capacity planning problems with an underlying lot sizing structure. Wolsey [462] uses the separation based reformulation method to generate a compact formulation for the dynamic lot size model with start-up costs.

We have studied four reformulations for the dynamic lot size model. The size of these reformulations is given in Table 16.1. It is interesting to summarize the definition of the “production” variables used in each model. In the original (*DLS*) model with  $(l, S)$  inequalities added the production variable is  $x_t$ , the quantity produced in period  $t$ . In the separation reformulation the auxiliary production variable  $z_{tk}$  can be interpreted as the amount of production in period  $t$  used to satisfy demand for periods  $k+1$  or greater. In the shortest route reformulation the auxiliary production variable  $z_{tk}$  is equal to one if there is production in period  $t$  to cover demand in periods  $t$  through  $k$ . Finally, in the multicommodity reformulation the auxiliary production variable  $z_{tk}$  can be interpreted as the amount of production in period  $t$  used to satisfy period  $k$  demand. The dynamic lot size model is truly a beautiful illustration of how important it is for a modeler to develop a variable definition that will lead to good linear programming relaxations!

This table suggests several interesting questions regarding mixed integer reformulations with auxiliary variables. First, for which mixed integer programming models do there exist compact formulations which are tight? Given that a compact reformulation in auxiliary variables exists, what is the minimum size reformulation in variables and constraints? The result of Yannakakis in the next section gives a lower bound on the number of rows in an extended polyhedral representation.

### 16.3 A PROJECTION THEOREM

Consider the polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ . Define the  $f \times \nu$  slack matrix  $M$  where element  $(i, j)$  is the slack of the  $i$ th facet of  $P$ ,  $i = 1, \dots, f$  evaluated at the  $j$ th extreme point of  $P$ ,  $j = 1, \dots, \nu$ .

**Theorem 16.23 (Yannakakis)** *The number of rows in any extended polyhedral representation of  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  cannot be less than  $m$ , where  $m$  is the smallest integer such that  $M = CD$ , and  $C$  and  $D$  are nonnegative matrices of dimension  $f \times m$  and  $m \times \nu$ , respectively.*

**Proof:**<sup>1</sup> Consider an arbitrary extended representation of  $Ax \geq b$  given by  $Fx + Gz \geq h$ , that is, projecting out the  $z$  variables from the system  $Fx + Gz \geq h$ , into the  $x$  variable subspace is the polyhedron  $P$  defined by the system  $Ax \geq b$ . Let  $m$  be the number of rows in the system  $Fx + Gz \geq h$ . From Theorem 2.22 in Chapter 2, the facets of  $P$  are contained among the constraints  $(u^i)^\top G \geq (u^i)^\top h$  where the  $u^i$  are extreme rays of the projection cone  $C_z = \{u \mid F^\top u = 0, u \geq 0\}$ . Let  $C$  be the matrix whose rows are defined by the  $(u^i)^\top$  which generate facets of  $P$ . Every extreme point  $x^j$  of  $P$  is the projection of an extreme point  $(x^j, z^j)$ . Then

$$\begin{aligned} M_{\bullet j} &= b - Ax^j \\ &= Ch - CFx^j - CGz^j \\ &= C(h - Fx^j - Gz^j). \end{aligned}$$

Observe that  $(h - Fx^j - Gz^j)$  is the  $j$ th column of the slack matrix of the reformulated model. Take  $D$  to be the slack matrix of the reformulated model. Then  $M = CD$ . The number of variables in the reformulated model is  $n$  plus the number of  $z$  variables and the number of rows is  $m$ . Since any such reformulation gives the desired matrix decomposition, the result is immediate.  $\square$

The key idea used here is that the slack matrix of the original system  $Ax \geq b$  is the projection matrix  $C$  times the slack matrix of the reformulated model.

### 16.4 BRANCH-AND-PRICE

It is quite possible that an extended formulation has too many auxiliary variables to handle if they are all included explicitly in the formulation. It may be

---

<sup>1</sup>Theorem 16.23 is due to Yannakakis [465]. This proof was communicated to me by R.G. Jeroslow on June 6, 1988.

necessary to use column generation with an extended formulation and price out the auxiliary variable columns implicitly by solving a subproblem. These ideas have been discussed in detail in Chapters 11 and 12. In the last several years, a number of researchers have worked directly with the formulation (*MIPFB*) which has an auxiliary variable for each of the finite generators of  $\text{conv}(\Gamma)$ . Typically, when solving the linear programming relaxation of (*MIPFB*) the constraints

$$x = \sum_{i=1}^q z_i x^i + \sum_{i=q+1}^r z_i x^i \quad (16.64)$$

are used to substitute out the  $x$  variables in the objective function  $c^\top x$ , and the coupling constraints  $Ax \geq b$ . Also, only a small subset of the remaining columns in (*MIPFB*) are used. Typically, the following restricted version of (*MIPFB*) is solved.

$$\begin{aligned} & \min \sum_{i \in \Lambda} (c^\top x^i) z_i \\ (RMIPFB) \quad & \text{s.t. } \sum_{i \in \Lambda} (Ax^i) z_i \geq b \\ & \sum_{i \in \Lambda} z_i = 1 \\ & z_i \geq 0, \quad i \in \Lambda \end{aligned}$$

Column generation is used to see if any columns not in  $\Lambda$  price out negative. Solving problem (*RMIPFB*), derived from (*MIPFB*) by using (16.64) to substitute out the  $x$  variables was discussed in Subsection 12.5.2 in Chapter 12. We illustrated the method with the dynamic lot sizing model. Until the 1990s subgradient optimization was almost always the tool of choice for optimizing the Lagrangian dual. Because of the great advances in linear programming technology in the 1990s, column generation is becoming much more viable, and perhaps superior to subgradient optimization. Another advantage of using linear programming combined with column generation is that it can easily be incorporated within a linear programming branch-and-bound algorithm. To understand branch-and-price, consider model (*MIPFB*). If  $x_j$  is an integer variable, and  $(\bar{x}, \bar{z})$  is a solution to the linear relaxation of (*MIPFB*) and  $\bar{x}_j$  is fractional, then by (16.64) the branch  $\bar{x}_j \geq \lceil \bar{x} \rceil$  is equivalent to

$$\sum_{i=1}^r z_i x_j^i = \bar{x}_j \geq \lceil \bar{x} \rceil. \quad (16.65)$$

Similarly for the branch  $\bar{x}_j \leq |\bar{x}|$ . One can then add the branching constraint (16.65) to the restricted formulation (*RMIPFB*) and perform column generation again. This is called *branch-and-price* and has been used by researchers such as Barnhart et al. [39], Barnhart, Hane, and Vance [37], Desrosiers et al. [120], Johnson [252], Savelsbergh [399], and Vance et al. [440]. The branching constraint is from Johnson [252]. See Barnhart et al. [39] for a good discussion of branching schemes that are structure dependent.

We illustrate this process with two examples. The first example is the generalized assignment problem and the second is the uncapacitated network flow problem. In the generalized assignment problem we use the knapsack constraints (capacity constraints) to define  $\Gamma$  and make use of the fact that when  $\Gamma = \cup_{k=1}^m \Gamma_k$  and  $\Gamma_i \cap \Gamma_j = \emptyset$  for  $i \neq j$  and the  $\Gamma_i$  are not identical sets it is more efficient to write (*RMIPFB*) as

$$\begin{aligned} \min & \sum_{k=1}^m \sum_{i \in \Lambda_k} (c^T x^{ki}) z_{ki} \\ \text{s.t. } & \sum_{k=1}^m \sum_{i \in \Lambda_k} (A^k x^{ki}) z_{ki} \geq b \\ & \sum_{i \in \Lambda_k} z_{ki} = 1, \quad k = 1, \dots, m \\ & z_{ki} \geq 0, \quad k = 1, \dots, m, \quad i \in \Lambda_k. \end{aligned}$$

See the discussion in Barnhart et al. [39] for the column generation approach in the presence of symmetry, that is, when the  $\Gamma_k$  and  $A^k$  are identical. An example of this is generalized assignment problem where each knapsack constraint has the same coefficients, i.e. each worker is identical.

Before beginning the examples, we note that when the coupling constraints constraints in (*MIP*) have the structure  $\sum_{k=1}^m x_{jk} = 1, j = 1, \dots, n$  as in the generalized assignment problem, then the formulation of (*MIPFB*) becomes

$$\begin{aligned} \min & \sum_{k=1}^m \sum_{j=1}^n \sum_{i \in \Lambda_k} (c_{jk} x_{jk}^i) z_{ki} \\ \text{s.t. } & \sum_{k=1}^m \sum_{i \in \Lambda_k} (x_{jk}^i) z_{ki} = 1, \quad j = 1, \dots, n \\ & \sum_{i \in \Lambda_k} z_{ki} = 1, \quad k = 1, \dots, m \\ & z_{ki} \geq 0, \quad k = 1, \dots, m, \quad i \in \Lambda_k \end{aligned}$$

where  $x_{jk}^i$  is the 0/1 value of variable  $x_{jk}$  in the  $i$ th 0/1 solution of the  $k$ th knapsack constraint  $\sum_{j=1}^n a_{jk}x_{jk} \leq b_k$ . Then the branching rule (16.65) is equivalent to the branching rule for the set partitioning problem based on Proposition 9.6 in Chapter 9. To see this, observe that if there is a fractional solution to (RMIPFB) then in at least one of the convexity constraints  $\sum_{i \in \Lambda_k} z_{ki} = 1$  there are two fractional  $\bar{z}_{ki}$  variables. By Proposition 9.6 there is an index  $s$  and index set  $I_{sk} = \{i \in \Lambda_k \mid x_{sk}^i = 1\}$  such that

$$0 < \sum_{i \in I_{sk}} \bar{z}_{ki} < 1.$$

Two new candidate problems are created by adding the constraints  $\sum_{i \in I_{sk}} z_{ki} = 1$  and  $\sum_{i \in I_{sk}} z_{ki} = 0$ . But by definition of  $I_{sk}$ ,  $\sum_{i \in I_{sk}} z_{ki} = x_{sk}$  so the this set partitioning based branching is equivalent to the branching rule (16.65) for the generalized assignment problem. It is also an example of the SOS 1 branching method developed in Chapter 9.

**Example 16.24** Consider the generalized assignment problem used in Example 16.18. The test problem is

$$\begin{aligned} \text{MIN } & 2 X_{11} + 11 X_{12} + 7 X_{21} + 7 X_{22} + 20 X_{31} + 2 X_{32} + 5 X_{41} + 5 X_{42} \\ \text{SUBJECT TO } & \\ 2) \quad & X_{11} + X_{12} = 1 \\ 3) \quad & X_{21} + X_{22} = 1 \\ 4) \quad & X_{31} + X_{32} = 1 \\ 5) \quad & X_{41} + X_{42} = 1 \\ 6) \quad & 3 X_{11} + 6 X_{21} + 5 X_{31} + 7 X_{41} \leq 13 \\ 7) \quad & 2 X_{12} + 4 X_{22} + 10 X_{32} + 4 X_{42} \leq 10 \end{aligned}$$

Initialize the master program with the following extreme point solutions to each of the knapsack constraints.

**Knapsack 1:**

$$\begin{aligned} x_{11}^1 &= 1, x_{21}^1 = 0, x_{31}^1 = 0, x_{41}^1 = 0 \\ x_{11}^2 &= 0, x_{21}^2 = 1, x_{31}^2 = 0, x_{41}^2 = 0 \\ x_{11}^3 &= 0, x_{21}^3 = 0, x_{31}^3 = 1, x_{41}^3 = 0 \\ x_{11}^4 &= 0, x_{21}^4 = 0, x_{31}^4 = 0, x_{41}^4 = 1 \end{aligned}$$

**Knapsack 2:**

$$\begin{aligned} x_{12}^1 &= 1, x_{22}^1 = 1, x_{32}^1 = 0, x_{42}^1 = 1 \\ x_{12}^2 &= 0, x_{22}^2 = 0, x_{32}^2 = 1, x_{42}^2 = 0 \end{aligned}$$

These extreme points lead to the following restricted master and solution.

```

MIN  2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 23 Z21 + 2 Z22
SUBJECT TO
 2)  Z11 + Z21 =    1
 3)  Z12 + Z21 =    1
 4)  Z13 + Z22 =    1
 5)  Z14 + Z21 =    1
 6)  Z11 + Z12 + Z13 + Z14 =    1
 7)  Z21 + Z22 =    1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 43.00000

VARIABLE	VALUE	REDUCED COST
Z13	1.000000	.000000
Z21	1.000000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-15.500000
3)	.000000	-20.500000
4)	.000000	-33.500000
5)	.000000	-18.500000
6)	.000000	13.500000
7)	.000000	31.500000

Generate a new column based on these dual prices. Refer back to Chapters 11 and 12 to see how the dual variables are used to generate the subproblem objective function. For the first knapsack the column generation problem is

$$\begin{aligned}
 \min \quad & -13.5x_{11} - 13.5x_{21} - 13.5x_{31} - 13.5x_{41} + 13.5 \\
 \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\
 & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}
 \end{aligned}$$

There are alternative optima to this subproblem and more than one column prices out negative. Two optimal solutions are

$$\begin{aligned}
 x_{11} = 1, x_{21} = 0, x_{31} = 1, x_{41} &= 0 \\
 x_{11} = 0, x_{21} = 1, x_{31} = 0, x_{41} &= 1
 \end{aligned}$$

The optimal objective function value is  $-13.5$  so columns corresponding to these solutions price out negative and must be added to the restricted master. This gives a new restricted master and solution.

```

MIN 2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 22 Z15 + 12 Z16 + 23 Z21 + 2 Z22
SUBJECT TO
 2) Z11 + Z15 + Z21 = 1
 3) Z12 + Z16 + Z21 = 1
 4) Z13 + Z15 + Z22 = 1
 5) Z14 + Z16 + Z21 = 1
 6) Z11 + Z12 + Z13 + Z14 + Z15 + Z16 = 1
 7) Z21 + Z22 = 1
END

```

## OBJECTIVE FUNCTION VALUE

1) 29.50000

VARIABLE	VALUE	REDUCED COST
Z15	.500000	.000000
Z16	.500000	.000000
Z21	.500000	.000000
Z22	.500000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-15.500000
3)	.000000	-7.000000
4)	.000000	-20.000000
5)	.000000	-18.500000
6)	.000000	13.500000
7)	.000000	18.000000

The optimal objective function value is reduced from 43 to 29.5. Price out the columns using the new dual prices. For the first knapsack the column generation problem is

$$\begin{aligned}
 \min \quad & -13.5x_{11} - 0x_{21} - 0x_{31} - 13.5x_{41} + 13.5 \\
 \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\
 & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}
 \end{aligned}$$

An optimal solution is  $x_{11} = 1$ ,  $x_{21} = 0$ ,  $x_{31} = 0$ , and  $x_{41} = 1$ . The optimal objective function value is  $-13.5$  so the column corresponding to this solution

prices out negative and must be added to the restricted master. This gives a new restricted master and solution.

```

MIN      2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 22 Z15 + 12 Z16 + 23 Z21
        + 2 Z22 + 7 Z17
SUBJECT TO
 2) Z11 + Z15 + Z21 + Z17 =    1
 3) Z12 + Z16 + Z21 =    1
 4) Z13 + Z15 + Z22 =    1
 5) Z14 + Z16 + Z21 + Z17 =    1
 6) Z11 + Z12 + Z13 + Z14 + Z15 + Z16 + Z17 =    1
 7) Z21 + Z22 =    1
END

```

## OBJECTIVE FUNCTION VALUE

1) 29.50000

VARIABLE	VALUE	REDUCED COST
Z15	.500000	.000000
Z16	.500000	.000000
Z21	.500000	.000000
Z22	.500000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-15.500000
3)	.000000	-20.500000
4)	.000000	-20.000000
5)	.000000	-5.000000
6)	.000000	13.500000
7)	.000000	18.000000

The optimal objective function value remains 29.5 due to degeneracy. However, there is a new dual solution. Price out the columns not in the restricted master. For the first knapsack the column generation problem is

$$\begin{aligned}
 \min \quad & -13.5x_{11} - 13.5x_{21} - 0x_{31} - 0x_{41} + 13.5 \\
 \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\
 & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}
 \end{aligned}$$

An optimal solution is  $x_{11} = 1$ ,  $x_{21} = 1$ ,  $x_{31} = 0$ , and  $x_{41} = 0$ . The optimal objective function value is -13.5 so the column corresponding to this solution

prices out negative and must be added to the restricted master. The restricted master and solution is

```

MIN      2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 22 Z15 + 12 Z16 + 23 Z21
        + 2 Z22 + 7 Z17 + 9 Z18
SUBJECT TO
 2) Z11 + Z15 + Z21 + Z17 + Z18 =    1
 3) Z12 + Z16 + Z21 + Z18 =    1
 4) Z13 + Z15 + Z22 =    1
 5) Z14 + Z16 + Z21 + Z17 =    1
 6) Z11 + Z12 + Z13 + Z14 + Z15 + Z16 + Z17 + Z18 =    1
 7) Z21 + Z22 =    1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 29.50000

VARIABLE	VALUE	REDUCED COST
Z15	.500000	.000000
Z16	.500000	.000000
Z21	.500000	.000000
Z22	.500000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-15.500000
3)	.000000	-20.500000
4)	.000000	-33.500000
5)	.000000	-18.500000
6)	.000000	27.000000
7)	.000000	31.500000

Again, there was a degenerate pivot and a new dual solution. For the first knapsack the column generation problem is

$$\begin{aligned}
 \min \quad & -13.5x_{11} - 13.5x_{21} - 13.5x_{31} - 13.5x_{41} + 27 \\
 \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\
 & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}
 \end{aligned}$$

An optimal solution is  $x_{11} = 1$ ,  $x_{21} = 1$ ,  $x_{31} = 0$ , and  $x_{41} = 0$ . The optimal solution value is 0 so there are no columns to add from the first knapsack problem. For the second knapsack problem the pricing problem is

$$\min -4.5x_{12} - 13.5x_{22} - 31.5x_{32} - 13.5x_{42} + 31.5$$

$$\text{s.t.} \quad 2x_{12} + 4x_{22} + 10x_{32} + 4x_{42} \leq 10 \\ x_{12}, x_{22}, x_{32}, x_{42} \in \{0, 1\}$$

An optimal solution is  $x_{12} = 0$ ,  $x_{22} = 0$ ,  $x_{32} = 1$ , and  $x_{42} = 0$ . The optimal solution value is 0 so there are no columns to add for the second knapsack problem. Enter the branching phase.

**Node 1:** The optimal solution in the auxiliary variables of

$$z_{15} = z_{16} = z_{21} = z_{22} = .5$$

and all other  $z_{ij}$  equal to zero corresponds to all of the original  $x_{ij}$  variables equal to 0.5. Branch on  $x_{11}$ . In the current restricted master this means adding the constraint  $z_{11} + z_{15} + z_{17} + z_{18} = 1$ . The new master (with an artificial variable added to guarantee feasibility) and solution is

```

MIN      2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 22 Z15 + 12 Z16 + 23 Z21
        + 2 Z22 + 7 Z17 + 9 Z18 + 1000 A
SUBJECT TO
 2)  Z11 + Z15 + Z21 + Z17 + Z18 =    1
 3)  Z12 + Z16 + Z21 + Z18 =    1
 4)  Z13 + Z15 + Z22 =    1
 5)  Z14 + Z16 + Z21 + Z17 =    1
 6)  Z11 + Z12 + Z13 + Z14 + Z15 + Z16 + Z17 + Z18 =    1
 7)  Z21 + Z22 =    1
 8)  Z11 + Z15 + Z17 + Z18 + A =    1
END

```

#### OBJECTIVE FUNCTION VALUE

1) 529.5000

VARIABLE	VALUE	REDUCED COST
Z15	.500000	.000000
Z16	.500000	.000000
Z21	.500000	.000000
Z22	.500000	.000000
A	.500000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	484.500000
3)	.000000	-520.500000
4)	.000000	-533.500000
5)	.000000	-518.500000

6)	.000000	1027.000000
7)	.000000	531.500000
8)	.000000	-1000.000000

For the first knapsack the column generation problem is

$$\begin{aligned} \min \quad & -513.5x_{11} - 513.5x_{21} - 513.5x_{31} - 513.5x_{41} + 1027 \\ \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\ & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\} \end{aligned}$$

Note that the objective coefficient on variable  $x_{11}$  comes from the computation  $-513.5 = 2 + 484.5 - 1000$  where 1000 is the value of the dual variable on the constraint  $z_{11} + z_{15} + z_{17} + z_{18} = 1$  used for branching. An optimal solution is  $x_{11} = 1$ ,  $x_{21} = 1$ ,  $x_{31} = 0$ , and  $x_{41} = 0$ . The optimal objective function value is 0 so there are no columns to add for the first knapsack problem. For the second knapsack problem the pricing problem is

$$\begin{aligned} \min \quad & 495.5x_{12} - 513.5x_{22} - 531.5x_{32} - 513.5x_{42} + 531.5 \\ \text{s.t.} \quad & 2x_{12} + 4x_{22} + 10x_{32} + 4x_{42} \leq 10 \\ & x_{12}, x_{22}, x_{32}, x_{42} \in \{0, 1\} \end{aligned}$$

An optimal solution is  $x_{12} = 0$ ,  $x_{22} = 1$ ,  $x_{32} = 0$ , and  $x_{42} = 1$ . The optimal objective function value is -513.5 so add the corresponding column to the restricted master corresponding to branch 1. The new master and solution is

```

MIN      2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 22 Z15 + 12 Z16 + 23 Z21
        + 2 Z22 + 7 Z17 + 9 Z18 + 1000 A + 12 Z23
SUBJECT TO
 2)  Z11 + Z15 + Z21 + Z17 + Z18 =      1
 3)  Z12 + Z16 + Z21 + Z18 + Z23 =      1
 4)  Z13 + Z15 + Z22 =      1
 5)  Z14 + Z16 + Z21 + Z17 + Z23 =      1
 6)  Z11 + Z12 + Z13 + Z14 + Z15 + Z16 + Z17 + Z18 =      1
 7)  Z21 + Z22 + Z23 =      1
 8)  Z11 + Z15 + Z17 + Z18 + A =      1
END

OBJECTIVE FUNCTION VALUE

1)      34.00000

VARIABLE      VALUE      REDUCED COST

```

Z15	1.000000	.000000
Z23	1.000000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
3)	.000000	-25.000000
4)	.000000	-38.000000
5)	.000000	-23.000000
6)	.000000	36.000000
7)	.000000	36.000000
8)	.000000	-20.000000

For the first knapsack the column generation problem is

$$\begin{aligned} \min \quad & 2x_{11} - 18x_{21} - 18x_{31} - 18x_{41} + 36 \\ \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\ & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\} \end{aligned}$$

An optimal solution is  $x_{21} = 0$ ,  $x_{31} = 1$ , and  $x_{41} = 0$ . The optimal objective function value is 0 and no columns are added from the first knapsack problem. For the second knapsack problem the pricing problem is

$$\begin{aligned} \min \quad & 11x_{12} - 18x_{22} - 36x_{32} - 18x_{42} + 36 \\ \text{s.t.} \quad & 2x_{12} + 4x_{22} + 10x_{32} + 4x_{42} \leq 10 \\ & x_{12}, x_{22}, x_{32}, x_{42} \in \{0, 1\} \end{aligned}$$

An optimal solution is  $x_{12} = 0$ ,  $x_{22} = 1$ ,  $x_{32} = 0$ , and  $x_{42} = 1$ . The optimal objective function value is 0 and no columns are added from the second knapsack problem. Therefore, the optimal solution in auxiliary variables at node 1 is  $z_{15} = 1$  and  $z_{23} = 1$ .

**Node 2:** Create node 2 by fixing  $x_{11} = 0$ . This corresponds to adding  $z_{11} + z_{15} + z_{17} + z_{18} = 0$  to the restricted master. The formulation and solution is

$$\begin{aligned} \text{MIN} \quad & 2 Z11 + 7 Z12 + 20 Z13 + 5 Z14 + 22 Z15 + 12 Z16 + 23 Z21 \\ & + 2 Z22 + 7 Z17 + 9 Z18 \end{aligned}$$

SUBJECT TO

$$\begin{aligned} 2) \quad & Z11 + Z15 + Z21 + Z17 + Z18 = 1 \\ 3) \quad & Z12 + Z16 + Z21 + Z18 = 1 \\ 4) \quad & Z13 + Z15 + Z22 = 1 \\ 5) \quad & Z14 + Z16 + Z21 + Z17 = 1 \\ 6) \quad & Z11 + Z12 + Z13 + Z14 + Z15 + Z16 + Z17 + Z18 = 1 \end{aligned}$$

```

7) Z21 + Z22 = 1
8) Z11 + Z15 + Z17 + Z18 = 0
END

```

## OBJECTIVE FUNCTION VALUE

1) 43.00000

VARIABLE	VALUE	REDUCED COST
Z13	1.000000	.000000
Z21	1.000000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-41.000000
4)	.000000	-20.000000
7)	.000000	18.000000
8)	.000000	39.000000

The first knapsack problem prices out to zero. The second knapsack problem prices out negative, but adding columns from the second knapsack problem only changes the dual solution and not the primal solution which remains optimal. This optimal solution is also integer so no further branching is necessary. Since there was a better objective function value, 34 versus 43, at node 1, the node 1 solution is optimal. The optimal solution in the original variable space is

$$x_{11} = x_{31} = x_{22} = x_{42} = 1$$

and all others 0 for an optimal solution value of 34.

**Example 16.25** The arc-path reformulation APUNF, of the uncapacitated network flow problem (UNF) generally has too many variables and they cannot all be carried explicitly in the model. The arc-path reformulation is an excellent candidate for branch-and-price. Barnhart, Hane, and Vance [37] apply branch-and-price to a version of APUNF where all of the flow from the origin vertex 0 to the destination vertex  $k \in D$  is required to follow a single a path. The pricing subproblem to generate columns is a shortest path problem which can be solved very efficiently.

## 16.5 PROJECTION OF EXTENDED FORMULATIONS: BENDERS' DECOMPOSITION REVISITED

There are two ways to use an extended polyhedral reformulation. One way is to use the extended formulation within a linear programming based branch-and-bound algorithm. Eppen and Martin [139], Jeroslow and Lowe [250], Martin [316], Rardin and Choe [376], and Sahinidis and Grossmann [390] report computational evidence with this approach where all of the auxiliary variables are explicitly carried in the formulation. Barnhart et al. [39], Barnhart, Hane, and Vance [37], and Savelsbergh [399] all report computational evidence with this approach but the auxiliary variables are generated within a branch-and-price algorithm. A second use of extended formulations is to generate valid cutting planes by projecting out the auxiliary variables. Balas [20, 22, 23] in his work on disjunctive programming and Balas and Pulleyblank [29] in their work on the perfectly matchable subgraph polytope of a bipartite graph were the first to do this. In this section we discuss several examples of this. First we apply Benders' decomposition to the general model reformulation (*RMIP*) and project out auxiliary variables in order to generate cutting planes on-the-fly. We then specialize the Benders' procedure to several important problem classes.

Apply Benders' decomposition to problem (*RMIP*) defined by (16.3)-(16.6). In model (*RMIP*) project out the auxiliary  $z$  variables. Let  $u^i$  be the dual multipliers on constraints  $Fx + Gz \geq h$ . The resulting projected system, or Benders' master program, is

$$\min \quad c^\top x \tag{16.66}$$

$$\text{s.t.} \quad Ax \geq b \tag{16.67}$$

$$(u^i)^\top Fx \geq (u^i)^\top h, \quad i = 1, \dots, t \tag{16.68}$$

$$x \geq 0 \tag{16.69}$$

$$x_j \in \mathbb{Z}, \quad j \in I \tag{16.70}$$

Recall that the idea behind Benders' decomposition is to solve a relaxed master with only a small subset of the constraints (16.68). Then solve a Benders' subproblem to see any of the constraints not present in the relaxed master are violated. If so, add a violated cut. The subproblem is formulated as follows. Let  $\bar{x}$  be a solution to the relaxed master. If there is a  $\bar{z}$  such that  $F\bar{x} + G\bar{z} \geq h$  then  $\bar{x} \in \text{conv}(\Gamma)$  and there are no violated cuts. However, if there is no such  $\bar{z}$ , i.e. there is no solution to the system  $Gz \geq h - F\bar{x}$ , then by Farkas' lemma

there is a solution to the system

$$(h - F\bar{x})^\top u > 0, \quad G^\top u = 0, \quad u \geq 0.$$

Finding a solution to this system is equivalent to solving the following linear program which is the Benders' subproblem.

$$\max \{(h - F\bar{x})^\top u \mid G^\top u = 0, u \geq 0\}. \quad (16.71)$$

If, indeed,  $\bar{x}$  is not contained in  $\text{conv}(\Gamma)$  then this linear program is unbounded and there is an extreme ray  $\bar{u}$  such that  $(h - F\bar{x})^\top \bar{u} > 0$ . This extreme ray provides a new cut  $(\bar{u}^\top F)x \geq \bar{u}^\top h$ .

Repeatedly solving (16.66)-(16.69) with even a small subset of the constraints (16.68) present as an integer program could be very time consuming. An alternative strategy is to solve the linear programming relaxation of (16.66)-(16.69) with a small subset of the constraints (16.68), add violated as needed, and then go into branch-and-bound, perhaps employing branch-and-cut. Also, in order to reduce the potential number of Benders' cuts, it makes sense to add the  $Bx \geq d$  constraints to the relaxed master. This linear programming relaxation is

$$\begin{aligned} & \min c^\top x \\ \text{s.t. } & Ax \geq b \\ & Bx \geq d \\ & (u^i)^\top Fx \geq (u^i)^\top h, \quad i = 1, \dots, k \leq t \\ & x \geq 0. \end{aligned}$$

The Benders' algorithm applied to this problem is

#### Algorithm 16.26 (Projection of Extended Formulation)

**Step 1: (Initialization)** Set  $k \leftarrow 0$  and create a relaxed master with only the  $Ax \geq b$ ,  $Bx \geq d$  and nonnegativity constraints.

**Step 2: (Relaxed Master Step)** Solve the current version of the relaxed master program as a linear program and obtain the solution  $x^k$ . If any of the integer variables are fractional go to **Step 3**. Otherwise terminate with the optimal integer solution.

**Step 3: (Subproblem Step)** Solve the subproblem  $\max \{(h - Fx^k)^\top u \mid G^\top u = 0, u \geq 0\}$ . If there is an optimal solution, the optimal solution value is 0. This

implies the solution to the relaxed master is the solution to the linear programming relaxation of the full master. Go to **Step 4**. Otherwise, the subproblem is unbounded and there is an extreme ray  $u^{k+1}$  such that  $(u^{k+1})^\top (h - Fx^k) > 0$ . Add the cut  $((u^{k+1})^\top F)x \geq (u^{k+1})^\top h$  and increment  $k \leftarrow k + 1$ . Go to **Step 2**.

**Step 4: (Branch-and-Bound)** Add the integrality constraints  $x_j$  integer for all  $j \in I$  and enter branch-and-bound if there are any fractional integer variables.

From a computational standpoint, it helps to make the subproblem  $\max \{(h - Fx^k)^\top u \mid G^\top u \leq 0, u \geq 0\}$ , which is a linear program over a cone, bounded. There are several ways to do this. First, one could add a constraint of the form  $\sum_{i=1}^m u_i \leq 1$ . The following lemma is left as Exercise 16.1.

**Lemma 16.27** If  $\bar{u}$  is an extreme point of the polytope

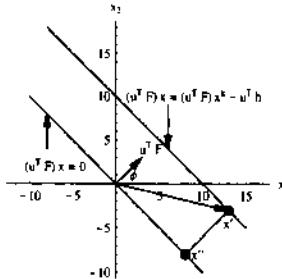
$$\{u \in \mathbb{R}^m \mid G^\top u = 0, \sum_{i=1}^m u_i \leq 1, u \geq 0\}$$

then  $\bar{u}$  is an extreme ray of the cone  $\{u \in \mathbb{R}^m \mid G^\top u = 0, u \geq 0\}$ .

Unfortunately, if  $\bar{u}$  is an extreme ray of the projection cone it does not follow that the corresponding cut is a facet of the projected polyhedron (See Balas [24] for conditions on when this is the case).

An alternative method is to find  $\bar{u}$  such that the corresponding cut is the “deepest possible.” By deepest possible, we mean a cut such that the distance between  $x^k$  and its orthogonal projection on the hyperplane defined by  $(u^\top F)x = u^\top h$  is maximized, that is the Euclidean distance between the hyperplanes  $(u^\top F)x = u^\top h$  and  $(u^\top F)x = (u^\top F)x^k$  is maximized. For ease of exposition, assume each hyperplane is translated by the amount  $-u^\top h$ . Let  $x'$  be the translated point  $x^k$  and  $x''$  its orthogonal projection on the hyperplane  $(u^\top F)x = 0$ . If  $H = \{x \mid (u^\top F)x = 0\}$  then  $H^\perp = \{x \mid x = \lambda(F^\top u)\}$  and there is a  $\bar{\lambda}$  such that  $x' = x'' + \bar{\lambda}(F^\top u)$ . Without loss, assume  $\|F^\top u\| = 1$ . Then, the Euclidean distance between  $x'$  and its orthogonal projection on  $H$  is

$$\begin{aligned} \|x' - x''\| &= \sqrt{(\bar{\lambda}u^\top F)(\bar{\lambda}F^\top u)} = |\bar{\lambda}| = \|x'\| \cos(\phi) = (u^\top F)x' \\ &= (u^\top F)x^k - u^\top h \end{aligned}$$

**Figure 16.6** Finding the Deepest Cut

where  $\phi$  is the angle between the vectors  $F^T u$  and  $x'$ . See Figure 16.6. Therefore, maximizing the Euclidean distance between  $x^k$  and its projection on the plane  $(u^T F)x = u^T h$  is given by the  $u$  which is the solution to the subproblem

$$\max \|x' - x''\| = \max \{(h - Fx^k)^T u \mid G^T u = 0, \|F^T u\| = 1, u \geq 0\}.$$

The feasible region is bounded however, this is a nonlinear problem and we choose to normalize by using the constraint  $\sum_{i=1}^m u_i \leq 1$ .

There is a row in the cut generation subproblem  $\max \{(h - Fx^k)^T u \mid G^T u \leq 0, \sum_{i=1}^m u_i = 1, u \geq 0\}$  for each auxiliary variable in the extended reformulation (RMIP). If there are many more auxiliary variables than constraints in the reformulated model (as there are in the example of the next subsection) then it is more efficient to solve the dual of the cut generation subproblem combined with column generation. The dual of the cut generation subproblem  $\max \{(h - Fx^k)^T u \mid G^T u = 0, u \geq 0, \sum_{i=1}^m u_i \leq 1\}$  is

$$\begin{array}{ll} \min & z_0 \\ \text{s.t.} & Gz + z_0 \geq h - Fx^k \\ & z, z_0 \geq 0 \end{array}$$

In the Benders' algorithm applied to (RMIP) we extend the Subproblem Step by applying column generation to this dual subproblem.

### Step 3: (Subproblem Step)

**Step 3a:** Assume that  $\Lambda$  has been initialized to index a subset of the columns in matrix  $G$ . Solve the restricted subproblem

$$\min \quad z_0$$

$$\text{s.t. } \sum_{i \in \Lambda} g^i z_i + z_0 \geq h - Fx^k \\ z, z_0 \geq 0$$

where  $g^i$  is column  $i$  of matrix  $G$ . Let  $u^{k+1}$  be the optimal dual variables on the constraints.

**Step 3b:** Price out the columns of  $G$  which are not in  $\Lambda$ . Do this by calculating  $-(u^{k+1})^\top g^i$  for every column  $g^i$  of  $G$  not indexed by  $\Lambda$ . If  $G$  has a huge number of rows this pricing step should be done implicitly, i.e. by solving a subproblem. This is illustrated in the next subsection. If one or more of these columns price out negative add them to  $\Lambda$  and go back to **Step 3a**. If all price out nonnegative there is an optimal dual solution  $u^{k+1}$  to the full subproblem. If  $z_0 = 0$  there are no more cuts. Go to **Step 4**. Otherwise, add the cut  $((u^{k+1})^\top F)x \geq (u^{k+1})^\top h$  and increment  $k \leftarrow k + 1$ . **Go to Step 2.**

These concepts are illustrated in the next three subsections.

### 16.5.1 Fenchel Cutting Planes

In this subsection we apply the Benders' algorithm to the extended formulation based on the integer finite basis theorem. The method we present is due to Boyd [66, 67, 68]. Boyd derives his cuts using Fenchel duality. See Rockafellar [382]. In keeping with the theme of this book, we derive the Fenchel cutting planes by projecting out the auxiliary variables of the extended formulation (*MIPFB*). The Benders' algorithm developed previously for the general reformulation (*RMIP*) applies directly to (*MIPFB*). The  $Fx + Gz \geq h$  constraints in (*RMIP*) correspond to the constraints

$$\begin{aligned} \sum_{i=1}^q x^i z_i + \sum_{i=q+1}^r x^i z_i - x &= 0 \\ \sum_{i=1}^q z_i &= 1 \\ z &\geq 0. \end{aligned}$$

In the Subproblem Step, the problem solved to generate a cut for a given  $\bar{x}$  is

$$\max \quad \bar{x}^T u + u_0$$

$$\begin{aligned} \text{s.t. } & (x^i)^\top u + u_0 \leq 0, \quad i = 1, \dots, q \\ & (x^i)^\top u \leq 0, \quad i = q+1, \dots, r \\ & u_0 \leq 0 \end{aligned}$$

where  $u_0$  is the dual variable on the convexity constraint  $\sum_{i=1}^q z_i = 1$ . Since the constraints with the auxiliary variables in (MIPFB) are strict equality constraints the  $u$  vector of dual multipliers is unrestricted in sign. So far this discussion has been independent of problem structure associated with (MIP). Now assume as in Boyd [67] that  $\Gamma = \{x \in \mathbb{R}^n \mid a^\top x \leq a_0, x_i \in \{0, 1\}, i = 1, \dots, n\}$  where  $a$  is a nonnegative  $n$  component vector and  $a_0$  is a positive scalar. Thus,  $\Gamma$  is the set of integer points defined by a single binary knapsack constraint. In this case  $\text{conv}(\Gamma)$  is polytope defined by all convex combinations of the binary vectors that satisfy  $a^\top x \leq a_0$ .

**Lemma 16.28** *Let  $x^i, i = 1, \dots, q$  denote the extreme points of  $\text{conv}(\Gamma)$  where  $\Gamma = \{x \in \mathbb{R}^n \mid a^\top x \leq a_0, x_i \in \{0, 1\}, i = 1, \dots, n\}$  and  $a, a_0$  are nonnegative. Then  $\bar{x} \in \text{conv}(\Gamma)$  if and only if the optimal solution value of*

$$\min \quad z_0 \tag{16.72}$$

$$\text{s.t. } \sum_{i=1}^q x^i z_i + z_0 e \geq \bar{x} \tag{16.73}$$

$$\sum_{i=1}^q z_i = 1 \tag{16.74}$$

$$z \geq 0 \tag{16.75}$$

is zero. Furthermore, if the optimal solution value of  $z_0$  is strictly positive and  $\bar{u}, \bar{u}_0$  are the optimal dual variables for constraints (16.73) and (16.74) respectively, then the cut  $\bar{u}^\top x + \bar{u}_0 \leq 0$  separates  $\bar{x}$  from  $\text{conv}(\Gamma)$ .

**Lemma 16.29** *If  $a$  is nonnegative then Lemma 16.28 is valid if all the rows in (16.73) corresponding to  $\bar{x}_j = 0$  are deleted.*

The proofs of Lemmas 16.28 and 16.29 are left as Exercises 16.2 and 16.3.

The number of auxiliary variables in (16.73) is potentially huge because there is one auxiliary variable for every integer vector in  $\Gamma$ . The discussion in the previous section concerning working with a restricted subproblem and using column generation applies. The structure of (16.72)-(16.75) is particularly amenable to

a mechanical pricing method as opposed to explicitly pricing out every auxiliary variable. Let  $\Lambda$  index a subset of the auxiliary variables and assume that a restriction of (16.72)-(16.75) is solved over  $\Lambda$ . Let  $\bar{u}, \bar{u}_0$  are the optimal dual variables for constraints (16.73) and (16.74) respectively, from the solution of the restriction. Solve the knapsack problem

$$\max \bar{u}^T x + \bar{u}_0 \quad (16.76)$$

$$\text{s.t.} \quad a^T x \leq a_0 \quad (16.77)$$

$$x_i \in \{0, 1\}, \quad i = 1, \dots, n. \quad (16.78)$$

If  $x^l$  is the optimal binary solution and  $\bar{u}^T x^l + \bar{u}_0 > 0$  then add  $l$  to  $\Lambda$  and resolve the new restricted subproblem. If  $\bar{u}^T x^l + \bar{u}_0 \leq 0$  then the optimal solution to the restricted subproblem is the optimal solution to the subproblem (16.72)-(16.75). Algorithm 16.26 specialized for the case when  $\Gamma$  is defined by a single knapsack constraint is

**Step 1: (Initialization)** Set  $k \leftarrow 0$  and create the relaxed master with only the  $Ax \geq b$ ,  $a^T x \leq a_0$  and nonnegativity constraints. Generate a subset of feasible solutions  $x^i$  for the knapsack constraint and index these solutions with  $\Lambda$ .

**Step 2: (Relaxed Master Step)** Solve the current version of the master program

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \geq b \\ & a^T x \leq a_0 \\ & (u^i)^T x + u_0^i \leq 0, \quad i = 1, \dots, k \leq t \\ & x \geq 0 \end{array}$$

and obtain the solution  $\bar{x}$ . If any of the integer variables are fractional go to the **Step 3**. Otherwise, terminate with the optimal integer solution.

### Step 3: (Subproblem Step)

**Step 3a:** Solve the restricted subproblem (16.72)-(16.75) over a subset of columns indexed by  $\Lambda$ . Let  $\bar{u}$  and  $\bar{u}_0$  be the optimal dual variables on constraints (16.73) and (16.74), respectively.

**Step 3b:** Price out the columns not contained in  $\Lambda$ . Do this by solving the knapsack problem (16.76)-(16.78). Let  $x^l$  be the optimal solution. If

$\bar{u}^T x^l + \bar{u}_0 > 0$  then add  $l$  to  $A$  and return to **Step 3a**. If  $\bar{u}^T x^l + \bar{u}_0 \leq 0$  then all elements in  $\Gamma$  price out nonnegative and  $\bar{u}, \bar{u}_0$  is an optimal dual solution of (16.72)-(16.75). If  $z_0 = 0$  there are no violated cuts. Go to **Step 4**. Otherwise, set  $u^{k+1} \leftarrow \bar{u}$ ,  $u_0^{k+1} \leftarrow \bar{u}_0$  add the cut  $(u^{k+1})^T x + u_0^{k+1} \leq 0$  to the relaxed master and increment  $k \leftarrow k + 1$ . Go to Step 2.

**Step 4: (Branch-and-Bound)** Solve the current version of the master program as an integer linear program.

We are really combining many key points for solving large scale problems such as projection, row generation and column generation.

If the  $Ax \geq b$  constraints in (*MIP*) contain knapsack constraints then we can apply this process to each constraint individually. Again, the philosophy from Chapter 15 applies, namely, if the problem's feasible region is

$$\Gamma = \{x \mid (a^i)^T x \leq a_{0i}, i = 1, \dots, m, x_j \in \{0, 1\}, j = 1, \dots, n\}$$

and  $\Gamma_i = \{x \mid (a^i)^T x \leq a_{0i}, x_j \in \{0, 1\}, j = 1, \dots, n\}$  then  $\bigcap_{i=1}^m \text{conv}(\Gamma_i)$  is a good approximation of  $\text{conv}(\Gamma)$  when the problem is very sparse. The computational results in Boyd [67] with the Crowder, Johnson and Padberg [103] test data set again confirm this idea. On three of the these test problems Boyd gets better bounds than Crowder, Johnson, and Padberg did using the methods developed in Chapter 15. This is because the auxiliary variable formulation using the finite basis theorem combined with projection is guaranteed to generate  $\text{conv}(\Gamma_i)$  and the minimal cover cuts are not a complete characterization of the convex hull of the binary knapsack. This phenomenon is illustrated in the next example.

**Example 16.30** Consider the generalized assignment problem from example 16.18. The test problem is

```

MIN    2 X11 + 11 X12 + 7 X21 + 7 X22 + 20 X31 + 2 X32 + 5 X41 + 5 X42
SUBJECT TO
 2)  X11 + X12 =      1
 3)  X21 + X22 =      1
 4)  X31 + X32 =      1
 5)  X41 + X42 =      1
 6)  3 X11 + 6 X21 + 5 X31 + 7 X41 <=   13
 7)  2 X12 + 4 X22 + 10 X32 + 4 X42 <=   10

```

The solution to the linear programming relaxation with all of the minimal cover cuts for the knapsack constraints 6) and 7) added is

$$x_{11} = 1, x_{21} = 0.5, x_{22} = 0.5, x_{31} = 0.5, x_{32} = 0.5, x_{41} = 0.5, x_{42} = 0$$

and the optimal solution value is 25.0. First generate cuts using the knapsack constraint

$$3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13.$$

Use the following four solutions to initialize the subproblem restricted master.

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

The subproblem solution is

```

MIN      Z0
SUBJECT TO
 2)  Z0 + Z1 >=  1
 3)  Z0 + Z2 >=  0.5
 4)  Z0 + Z3 >=  0.5
 5)  Z0 + Z4 >=  0.5
 6)  Z1 + Z2 + Z3 + Z4 =     1
END

```

#### OBJECTIVE FUNCTION VALUE

1) .3750000

VARIABLE	VALUE	REDUCED COST
Z0	.375000	.000000
Z1	.625000	.000000
Z2	.125000	.000000
Z3	.125000	.000000
Z4	.125000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	-.000000	-.250000
3)	-.000000	-.250000
4)	-.000000	-.250000
5)	-.000000	-.250000
6)	.000000	.250000

Now do column generation. Solve the knapsack problem

$$\max (1/4)x_{11} + (1/4)x_{21} + (1/4)x_{31} + (1/4)x_{41} - (1/4)$$

s.t.

$$3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13$$

$$x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}$$

*There are alternative optima. Take two of the solutions.*

$$x_{11} = 1, x_{21} = 0, x_{31} = 1, x_{41} = 0$$

$$x_{11} = 0, x_{21} = 1, x_{31} = 0, x_{41} = 1$$

*Both solutions price out at (1/4) and must be added to the subproblem linear program.*

```

MIN      Z0
SUBJECT TO
 2)  Z0 + Z1 + Z5 >= 1
 3)  Z0 + Z2 + Z6 >= 0.5
 4)  Z0 + Z5 + Z3 >= 0.5
 5)  Z0 + Z6 + Z4 >= 0.5
 6)  Z1 + Z5 + Z2 + Z6 + Z3 + Z4 = 1
END

```

#### OBJECTIVE FUNCTION VALUE

1) .2500000

VARIABLE	VALUE	REDUCED COST
Z0	.250000	.000000
Z5	.750000	.000000
Z6	.250000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	-.000000	-.500000
5)	-.000000	-.500000
6)	.000000	.500000

NO. ITERATIONS= 3

*Do column generation again. Solve the knapsack problem*

$$\max (1/2)x_{11} + 0x_{21} + 0x_{31} + (1/2)x_{41} - (1/2)$$

s.t.

$$3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13$$

$$x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}$$

The optimal solution is  $x_{11} = 1, x_{21} = 0, x_{31} = 0, x_{41} = 1$  for an optimal objective function value of  $(1/2)$ . Therefore this extreme point corresponds to a negative reduced cost and the corresponding column is added to the subproblem linear program.

```

MIN      Z0
SUBJECT TO
 2)  Z0 + Z1 + Z5 + Z7 >=  1
 3)  Z0 + Z2 + Z6 >=  0.5
 4)  Z0 + Z5 + Z3 >=  0.5
 5)  Z0 + Z7 + Z6 + Z4 >=  0.5
 6)  Z1 + Z5 + Z7 + Z2 + Z6 + Z3 + Z4 =  1
END

```

#### OBJECTIVE FUNCTION VALUE

1) .2500000

VARIABLE	VALUE	REDUCED COST
Z0	.250000	.000000
Z5	.750000	.000000
Z6	.250000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	-.000000	-.500000
3)	-.000000	-.500000
6)	.000000	.500000

There was a degenerate pivot and no progress was made in reducing the primal. The dual solution has changed so do column generation again. Solve the knapsack problem

$$\begin{aligned}
 \max \quad & (1/2)x_{11} + (1/2)x_{21} + 0x_{31} + 0x_{41} - (1/2) \\
 \text{s.t.} \quad & 3x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 13 \\
 & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\}
 \end{aligned}$$

The optimal solution is  $x_{11} = x_{21} = 1, x_{31} = x_{41} = 0$  for an optimal objective function value of  $(1/2)$ . This solution corresponds to a negative reduced cost and the corresponding column is added to the subproblem linear program.

```
MIN      Z0
```

SUBJECT TO

- 2)  $Z_0 + Z_1 + Z_5 + Z_7 + Z_8 \geq 1$
- 3)  $Z_0 + Z_8 + Z_2 + Z_6 \geq 0.5$
- 4)  $Z_0 + Z_5 + Z_3 \geq 0.5$
- 5)  $Z_0 + Z_7 + Z_6 + Z_4 \geq 0.5$
- 6)  $Z_1 + Z_5 + Z_7 + Z_8 + Z_2 + Z_6 + Z_3 + Z_4 = 1$

END

OBJECTIVE FUNCTION VALUE

$$1) .1250000$$

VARIABLE	VALUE	REDUCED COST
$Z_0$	.125000	.000000
$Z_5$	.375000	.000000
$Z_7$	.250000	.000000
$Z_8$	.250000	.000000
$Z_6$	.125000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	-.000000	-.250000
3)	-.000000	-.250000
4)	-.000000	-.250000
5)	-.000000	-.250000
6)	.000000	.500000

*Do column generation again. Solve the knapsack problem*

$$\begin{aligned} \max \quad & (1/4)x_{11} + (1/4)x_{21} + (1/4)x_{31} + (1/4)x_{41} - (1/4) \\ \text{s.t.} \quad & 2x_{11} + 6x_{21} + 5x_{31} + 7x_{41} \leq 11 \\ & x_{11}, x_{21}, x_{31}, x_{41} \in \{0, 1\} \end{aligned}$$

*The optimal objective function value is zero. Therefore all the reduced costs of the subproblem are nonnegative and the current solution to the subproblem is optimal. Since the optimal solution to the subproblem is positive we add a cut. Reading off the dual prices the implied Fenchel cut is (after multiplying by 4)*

$$x_{11} + x_{21} + x_{31} + x_{41} \leq 2$$

*At this point there is the option of trying to generate a cut based on the second knapsack or adding this cut to the current formulation and resolving the linear programming relaxation. We do the latter. The new formulation, including the minimal cover cuts, and solution is*

```

MIN      2 X11 + 11 X12 + 7 X21 + 7 X22 + 20 X31 + 2 X32 + 5 X41 + 5 X42
SUBJECT TO
 2)  X11 + X12 =    1
 3)  X21 + X22 =    1
 4)  X31 + X32 =    1
 5)  X41 + X42 =    1
 6)  3 X11 + 6 X21 + 5 X31 + 7 X41 <=   13
 7)  2 X12 + 4 X22 + 10 X32 + 4 X42 <=   10
 8)  X11 + X21 + X41 <=   2
 9)  X32 + X42 <=   1
10)  X11 + X31 + X41 <=   2
11)  X22 + X32 <=   1
12)  X11 + X21 + X31 + X41 <=   2
END

```

## OBJECTIVE FUNCTION VALUE

1) 29.50000

VARIABLE	VALUE	REDUCED COST
X11	.500000	.000000
X12	.500000	.000000
X21	.500000	.000000
X22	.500000	.000000
X31	.500000	.000000
X32	.500000	.000000
X41	.500000	.000000
X42	.500000	.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	.000000	-15.500000
3)	.000000	-20.500000
4)	.000000	-33.500000
5)	.000000	-18.500000
7)	.000000	2.250000
9)	.000000	4.500000
11)	.000000	4.500000
12)	.000000	13.500000

Adding the Fenchel cut improves the linear programming relaxation value from 25 to 29.5 which is equal to the value of the linear programming relaxation of (MIPFB) which we found through column generation. We leave it to the reader to verify that this linear programming solution is in the convex hull of

solutions to both of the knapsack constraints. It is now necessary to enter the branch-and-bound phase.

### 16.5.2 Projection of Disjunctive Formulations

This subsection is based primarily on the work of Balas, Ceria, and Cornuéjols [25, 27]. For similar approaches see Sherali and Adams [407] and Lovász and Schrijver [295].

In the subsection on disjunctive programming we showed that the linear programming relaxation of  $(MIP)$  can be tightened by selecting an integer variable,  $x_k$ , (we assume here that all the integer variables are binary) and observing that  $\text{conv}(\Gamma_1 \cup \Gamma_2) \subseteq \bar{\Gamma}$  where

$$\begin{aligned}\bar{\Gamma} &= \{x \mid Ax \geq b, Bx \geq d, x \geq 0, x_j \leq 1, j \in I\} \\ \Gamma_1 &= \{x \mid Ax \geq b, Bx \geq d, x \geq 0, x_j \leq 1, j \in I, x_k = 1\} \\ \Gamma_2 &= \{x \mid Ax \geq b, Bx \geq d, x \geq 0, x_j \leq 1, j \in I, x_k = 0\}.\end{aligned}$$

The polyhedron  $\bar{\Gamma}$  is the feasible region of the linear programming relaxation of  $(MIP)$ . In many interesting cases  $\text{conv}(\Gamma_1 \cup \Gamma_2)$  is a proper subset of  $\bar{\Gamma}$ . By Proposition 16.16 an extended polyhedral description of  $\text{conv}(\Gamma_1 \cup \Gamma_2)$  is

$$(DMIP_k) \quad \begin{array}{lll} \min & c^\top x & \\ \text{s.t.} & \begin{array}{ll} Az^1 \geq bz_{10} & \alpha^1 \\ Az^2 \geq bz_{20} & \alpha^2 \\ Bz^1 \geq dz_{10} & \beta^1 \\ Bz^2 \geq dz_{20} & \beta^2 \\ z^1 + z^2 - x = 0 & u \\ z_{1j} \leq z_{10}, j \in I, v^1 \\ z_{2j} \leq z_{20}, j \in I, v^2 \\ z_{1k} - z_{10} = 0 & \theta_1 \\ z_{2k} = 0 & \theta_0 \\ z_{10} + z_{20} = 1 & u_0 \\ z^1, z^2, z_{10}, z_{20} \geq 0 & \end{array} \end{array}$$

Following the philosophy of this section, rather than solve the extended formulation  $(DMIP_k)$  directly, we apply Benders' decomposition and generate cuts. If  $x = \bar{x}$ , by Farkas' Lemma there is a feasible solution to  $(DMIP_k)$  if and only

if the optimal solution to the following linear program is zero.

$$(DISCUT) \quad \begin{aligned} \max \quad & u^T \bar{x} + u_0 \\ \text{s.t.} \quad & A^T \alpha^1 + B^T \beta^1 + \theta_1 e^k + u + v^1 \leq 0 \\ & A^T \alpha^2 + B^T \beta^2 + \theta_0 e^k + u + v^2 \leq 0 \\ & u_0 - b^T \alpha^1 - d^T \beta^1 - \theta_1 + e^T v^1 \leq 0 \\ & u_0 - b^T \alpha^2 - d^T \beta^2 + e^T v^2 \leq 0 \\ & \alpha^1, \alpha^2, \beta^1, \beta^2 \geq 0 \end{aligned}$$

This is a linear program over a cone and once again it is desirable to normalize the formulation to prevent an unbounded solution. Because  $u$  is unrestricted in sign the constraint  $\sum_{i=1}^n u_i \leq 1$  does not work. Instead, simply add simple upper and lower bounds of  $\pm 1$  on  $u$ . This gives the following cutting plane algorithm.

#### Algorithm 16.31 (Disjunctive-Projection Cutting Plane Algorithm)

**Step 1: (Initialization)** Initialize  $t \leftarrow 1$  and  $\Gamma_t = \{x \mid Ax \geq b, Bx \geq d, x \geq 0, x_j \leq 1 \text{ } j \in I\}$ .

**Step 2: (LP Solve)** Solve the linear program  $\min \{c^T x \mid x \in \Gamma_t\}$  and obtain solution  $x^t$ . If  $x_j^t \in \{0, 1\}$  for all  $j \in I$ , stop;  $x^t$  is the optimal solution to (MIP). Otherwise, go to **Step 3**.

**Step 3: (Variable Selection)** Select a fractional variable  $x_k^t$  where  $k \in I$ . Solve the corresponding linear programming problem (DISCUT) and obtain the cut  $u^T x + u_0 \leq 0$ . Go to **Step 4**.

**Step 4: (Add Cut)** Add the cut  $u^T x + u_0 \leq 0$  to the set of inequalities defining  $\Gamma_t$ . Increment  $t \leftarrow t + 1$  and go to **Step 2**.

An interesting feature of this algorithm is that after each cut is generated a new formulation with auxiliary variables is generated. This was not the case with the Fenchel cutting plane algorithm. Balas, Ceria, and Cornuéjols [25] show that the disjunctive approach outlined here is equivalent to selecting the index  $k \in I$ , multiplying the inequalities defining  $\Gamma_t$  by  $x_k$  and then linearizing by introducing new variables. Sherali and Adams [407] and Lovász and Schrijver [295] also construct their reformulations based on similar ideas. For computational results with the disjunctive-projection cutting plane algorithm see Balas, Ceria, and G. Cornuéjols [25]. In Balas, Ceria, and Cornuéjols [27], results with this cutting plane used in a branch-and-cut framework are given.

**Example 16.32** Refer back to the generalized assignment problem used in Example 16.18. Below is the linear program (DISCUT) used to find a cut based on projecting out the auxiliary variables from ( $DMIP_k$ ). As in Example 16.18, variable  $x_{41}$  was used for the disjunction. Also, the constraints  $z_{1j} \leq z_{10}$  and  $z_{2j} \leq z_{20}$  were not used since the  $\sum_{j=1}^2 x_{ij} = 1$  made the simple upper bounds on the binary variables unnecessary.

```

MAX      U11 + U21 + 0.24 U31 + 0.76 U32 + 0.4 U41 + 0.6 U42 + U0
SUBJECT TO
 2)  U11 + ALPHA11 - 3 BETA11 <= 0
 3)  ALPHA11 - 2 BETA12 + U12 <= 0
 4)  U21 - 6 BETA11 + ALPHA12 <= 0
 5) - 4 BETA12 + ALPHA12 + U22 <= 0
 6)  U31 - 5 BETA11 + ALPHA13 <= 0
 7)  U32 - 10 BETA12 + ALPHA13 <= 0
 8)  U41 - 7 BETA11 + ALPHA14 + THETA1 <= 0
 9)  U42 - 4 BETA12 + ALPHA14 <= 0
10)  U11 + ALPHA21 - 3 BETA21 <= 0
11)  U12 + ALPHA21 - 2 BETA22 <= 0
12)  U21 - 6 BETA21 + ALPHA22 <= 0
13)  U22 - 4 BETA22 + ALPHA22 <= 0
14)  U31 - 5 BETA21 + ALPHA23 <= 0
15)  U32 - 10 BETA22 + ALPHA23 <= 0
16)  U41 - 7 BETA21 + ALPHA24 + THETA0 <= 0
17)  U42 - 4 BETA22 + ALPHA24 <= 0
18)  U0 - ALPHA11 + 13 BETA11 + 10 BETA12 - ALPHA12 - ALPHA13
- ALPHA14 - THETA1 <= 0
19)  U0 - ALPHA21 + 13 BETA21 + 10 BETA22 - ALPHA22 - ALPHA23
- ALPHA24 <= 0

```

## OBJECTIVE FUNCTION VALUE

1) .6400000

VARIABLE	VALUE	REDUCED COST
U11	.500000	.000000
U21	1.000000	-.100000
U31	-1.000000	.120000
U32	1.000000	-.120000
U41	1.000000	0.000000
U42	-.633333	.000000
U0	-1.400000	.000000
ALPHA11	1.600000	.000000
BETA11	.700000	.000000
BETA12	.550000	.000000

U12	-1.000000	.000000
ALPHA12	3.200000	.000000
U22	-1.000000	.300000
ALPHA13	4.500000	.000000
THETA1	3.900000	.000000
BETA21	.166667	.000000
BETA22	.283333	.000000
ALPHA23	1.833333	.000000
ALPHA24	1.766667	.000000
THETAO	-1000.000000	.000000

The implied cut is

$$.5x_{11} - x_{12} + x_{21} - x_{22} - x_{31} + 1.0x_{32} + x_{41} - .6333333x_{42} \leq 1.4.$$

Unlike the Fenchel cuts which are based on each individual knapsack polytope, this cut involves variables from both of the knapsack constraints. When this cut is added and the linear program resolved the new objective function value is 21.4. This is the same value from the disjunctive formulation with auxiliary variables. Many implementation details and strategies of this projection method are in Balas, Ceria, and G. Cornuéjols [27].

### 16.5.3 Projection of Multi-commodity Formulation

The multicommodity reformulation is not an extended polyhedral representation of the convex hull of solutions to the uncapacitated network flow problem (*UNF*). However, in general, the linear programming relaxation of (*MCUNF*) is much stronger than the linear programming relaxation of (*UNF*). Unfortunately, the multicommodity reformulation can be quite large. Therefore, it is of interest to generate cuts by projecting out the auxiliary variables. Rardin and Wolsey [377] have given a complete characterization, *dicut collection cuts*, of the projection of (*MCUNF*) into the space of the original space ( $x, y$ ) variables. Unfortunately, no general separation algorithm for the general dicut inequality other than solving a linear program of the same size as (*MUNF*) is known. However, certain classes of valid inequalities, for example those of Van Roy and Wolsey [438] discussed in the previous chapter, are special cases of the dicut collection inequalities and have very efficient separation algorithms.

## 16.6 CONCLUSION

In this chapter we have shown the importance of generating alternative or extended formulations using auxiliary variables. An extended formulation in auxiliary variables can be used in two ways. First, one can use the extended formulation within a linear programming based branch-and-bound algorithm. One can either solve linear programming relaxation of the complete model with all of the auxiliary variables present as done by Eppen and Martin [139], Jeroslow and Lowe [250], Martin [316], Rardin and Choe [376], and Sahinidis and Grossmann [390]; or, one can generate auxiliary variables on-the-fly with a column generation method and continue to do so at each branch (branch-and-price). See, for example, Barnhart et al. [39], Barnhart, Hane, and Vance [37], and Savelsbergh[399].

The second approach with auxiliary variables is to apply Benders' decomposition to the extended formulation, project out the auxiliary variables and generate cuts in the original variable space. This is the approach taken by Balas, Ceria, and G. Cornuéjols [25, 27], Boyd [67], Balas and Pulleyblank [29], and Sherali and Adams [407]. Which of the two methods is more effective obviously depends upon the problem structure and implementation details. One thing that is significant about the approach of Balas, Ceria, and Cornuéjols [25, 27] is that it does not depend on the structure of (*MIP*). Their procedure will generate valid cuts for any integer program with 0/1 variables.

## 16.7 EXERCISES

- 16.1 Prove Lemma 16.27.
- 16.2 Prove Lemma 16.28.
- 16.3 Prove Lemma 16.29.
- 16.4 Is it valid to delete a row in the subproblem (16.72)-(16.75) that corresponds to  $\bar{x}_j = 1$ ? Why or why not?
- 16.5 Explain why no value of big M will work in the loose formulation (16.49)-(16.51) for the polyhedra of Example 16.19.
- 16.6 Prove Proposition 16.16.
- 16.7 Prove Proposition 16.17.

16.8 Why is Proposition 16.16 not valid if one of the polyhedrons is empty?

16.9 Redo Example 16.18 using variable  $x_{32}$ .

16.10 Two reformulations of  $(UNF)$  with auxiliary variables are  $(MCUNF)$  and  $(APUNF)$ . Discuss the advantages and disadvantage of each reformulation. In comparing the two formulations also consider the capacitated version of  $(UNF)$  where there are flow capacities on some of the arcs.

16.11 Formulate the simple or uncapacitated plant location problem as an uncapacitated network flow problem  $(UNF)$ . Then reformulate the problem as a multicommodity network flow problem  $(MCUNF(x, y, z))$ . In Chapter 15 we discussed the aggregated and disaggregated formulations of simple plant location. Show that formulation  $(UNF)$  corresponds to the aggregated formulation and that the multicommodity flow reformulation corresponds to the multicommodity reformulation.

16.12 Consider the dynamic lot sizing model with backlogging allowed. Let the variable  $w_t$  represent the amount of unsatisfied demand at the end of period  $t$ . The balance equations for each period are  $I_{t-1} + x_t + w_t - I_t = d_t + w_{t-1}$ . If  $T$  is the last period in the planning horizon then  $w_T = 0$  so all demand is satisfied. Formulate this problem as an uncapacitated network flow problem and as a multicommodity network flow problem. Give an example illustrating that the multicommodity uncapacitated network flow reformulation is not an extended formulation of the convex hull of solutions to the dynamic lot sizing model with backlogging allowed.

16.13 Show that the polyhedron defined by (16.7)-(16.11) has fractional extreme points.

16.14 Suppose  $\Gamma = \cup_{i=1}^{n-1} \Gamma_i$  where

$$\begin{aligned}\Gamma_i &= \{(x_1, \dots, x_n) \mid x_i + x_{i+1} = 1, x_i, x_{i+1} \geq 0, \\ &x_k = 0, k = 1, \dots, n, k \neq i, k \neq i+1\}.\end{aligned}$$

Give an extended polyhedral representation of  $\text{conv}(\Gamma)$  using the disjunctive method of this chapter.

16.15 Prove Proposition 16.5.

16.16 Give a multicommodity reformulation for the traveling salesman problem.

16.17 If  $\Gamma_1$  and  $\Gamma_2$  are nonempty polyhedra is  $\text{conv}(\Gamma_1 \cup \Gamma_2)$  a polyhedron?

- 16.18 Construct the send-and-split diagram (see Figure 16.3) for a five period dynamic lot sizing solution corresponding to  $x_1 = d_1 + d_2 + d_3$ ,  $x_2 = 0$ ,  $x_3 = 0$ ,  $x_4 = d_4$  and  $x_5 = d_5$ .
- 16.19 Prove Proposition 16.11.
- 16.20 Prove Lemma 16.13.
- 16.21 Two methods have been proposed for modeling piecewise linear functions. One method is using the special ordered sets (16.7)-(16.11). The other model is given in (9.3)-(9.5). Which method is better from an implementation standpoint?
- 16.22 Construct an extreme point solution of (16.40)-(16.43) for the graph of Figure 16.5 which has the property that at least one component of  $y$  is greater than 1.0.
- 16.23 Prove Proposition 16.12.

## **PART VI**

---

## **APPENDIX**

# A

---

## POLYHEDRAL THEORY

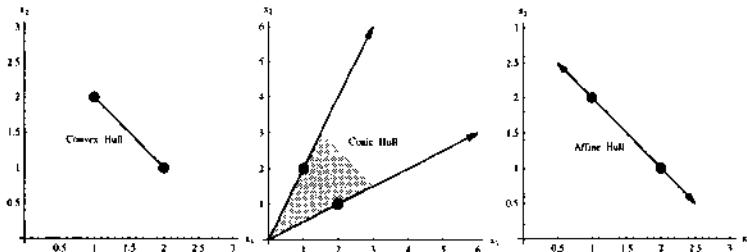
### A.1 INTRODUCTION

The main topics of this book are linear and integer linear programming. The properties of the feasible region of a linear program and the convex hull (defined in the next section) of the feasible region of an integer linear program are examined in this Appendix. This Appendix provides the reader with necessary background in polyhedral theory used throughout the text. A number of the results in this Appendix are easily proved by using linear programming or duality theory. However, we wish to make this Appendix independent of the rest of the book and hence make no use of linear programming theory. In Section A.2 key concepts used throughout the book are defined. Sections A.3 and A.4 are devoted to characterizing the feasible region of a linear program. In Section A.5 some necessary material about linear spaces is provided. Exercises are provided in Section A.6. For more material on convexity and polyhedral theory we refer the reader to Brøndsted [75], Rockafellar [382], and Stoer and Witzgall [413].

### A.2 CONCEPTS AND DEFINITIONS

The reader should be familiar with the concepts of a vector space, linear combination, and linear independence. These concepts from linear algebra are related to the following which are central to the study of linear optimization.

An *affine combination* of the points  $x^1, x^2, \dots, x^k$  in  $\mathbb{R}^n$  is a linear combination  $\lambda_1 x^1 + \lambda_2 x^2 + \dots + \lambda_k x^k$  such that  $\sum_{i=1}^k \lambda_i = 1$ .

**Figure A.1** Polyhedral Concepts

A *conic combination* of the points  $x^1, \dots, x^k$  in  $\mathbb{R}^n$  is a linear combination  $\lambda_1 x^1 + \dots + \lambda_k x^k$  such that  $\lambda_i \geq 0$  for all  $i = 1, \dots, k$ .

A *convex combination* of the points  $x^1, \dots, x^k$  in  $\mathbb{R}^n$  is a linear combination  $\lambda_1 x^1 + \dots + \lambda_k x^k$  such that  $\sum_{i=1}^k \lambda_i = 1$ , and  $\lambda_i \geq 0$  for all  $i = 1, \dots, k$ .

The *affine hull* of the points  $x^1, \dots, x^k$  is the set of points in  $\mathbb{R}^n$  formed by taking all affine combinations of these points. Similarly for *conic hull* and *convex hull*. The affine, conic and convex hull of the set of points  $\{x^1, \dots, x^k\}$  in  $\mathbb{R}^n$  are denoted by  $\text{aff}(\{x^1, \dots, x^k\})$ ,  $\text{cone}(\{x^1, \dots, x^k\})$ , and  $\text{conv}(\{x^1, \dots, x^k\})$ , respectively. The geometry of these concepts is illustrated in Figure A.1. This figure illustrates the convex, conic, and affine hull for two points in  $\mathbb{R}^2$ .

Similar to the concept of linear independence we define the concept of *affine independence*. The points  $x^1, \dots, x^k$  in  $\mathbb{R}^n$  are *affinely independent* if and only if none of the points is an affine combination of the remaining points.

If the affine combination of any set of points in  $A$  is also in  $A$ , then  $A$  is an *affine subspace* of  $\mathbb{R}^n$ . The *dimension* of an affine subspace  $A$  is  $n - 1$  if and only if  $n$  is the largest non-negative integer such that there are  $n$  affinely independent points in  $A$ . When  $A = \emptyset$  we say that  $A$  has dimension  $-1$ . The set  $C \subseteq \mathbb{R}^n$  is a *convex set* if the convex combination of every subset of points in  $C$  is also in  $C$ . If  $C$  is a convex set then  $\dim(C) = \dim(\text{aff}(C))$ , that is, the dimension of a convex set is the dimension of the affine hull of that convex set. Why is the concept of affine independence used to define dimension rather than linear independence? If the affine subspace  $A$  has dimension  $n$  does it contain  $n$  linearly independent points?

**Proposition A.1** *The vectors  $x^1, \dots, x^k$  are affinely independent if and only if  $(x^1 - x^j), \dots, (x^{j-1} - x^j), (x^{j+1} - x^j), \dots, (x^k - x^j)$  are linearly independent for  $j = 1, \dots, k$ .*

**Proof:** Without loss, let  $x^j = x^1$ . First show that the affine independence of  $x^1, \dots, x^k$  implies the linear independence of  $(x^2 - x^1), (x^3 - x^1), \dots, (x^k - x^1)$ . Prove the contrapositive and assume  $(x^2 - x^1), (x^3 - x^1), \dots, (x^k - x^1)$  are not linearly independent. Then there exists  $\lambda_2, \dots, \lambda_k$  not all zero such that

$$\sum_{i=2}^k \lambda_i (x^i - x^1) = 0.$$

Case i)  $\sum_{i=2}^k \lambda_i = 0$ . This implies  $\sum_{i=2}^k \lambda_i x^i = 0$ . Now assume without loss that  $\lambda_2 \neq 0$ . Then

$$x^2 = (-1/\lambda_2) \sum_{i=3}^k \lambda_i x^i.$$

But,  $\sum_{i=2}^k \lambda_i = 0$  implies  $\sum_{i=3}^k (\lambda_i/\lambda_2) = -1$  so we do not have affine independence.

Case ii)  $\sum_{i=2}^k \lambda_i \neq 0$ . Then

$$x^1 = \sum_{i=2}^k \theta_i x^i, \quad \theta_i = \lambda_i / \sum_{i=2}^k \lambda_i$$

again contradicting affine independence.

Next show if  $x^1, \dots, x^k$  are not affinely independent then  $(x^2 - x^1), (x^3 - x^1), \dots, (x^k - x^1)$  are not linearly independent. By definition of affine independence at least one of the  $x^i$  can be written as an affine combination of the others. Assume without loss, that this is  $x^2$ .

$$x^2 = \sum_{\substack{i=1 \\ i \neq 2}}^k \lambda_i x^i, \quad \sum_{\substack{i=1 \\ i \neq 2}}^k \lambda_i = 1.$$

Then

$$\sum_{\substack{i=1 \\ i \neq 2}}^k \lambda_i (x^i - x^2) = 0$$

which implies the vectors  $(x^1 - x^2), (x^3 - x^2), \dots, (x^k - x^2)$  are not linearly independent since not all  $\lambda_i$  are zero. Multiply  $(x^1 - x^2)$  by -1 and add it to  $(x^3 - x^2), \dots, (x^k - x^2)$  and observe that  $(x^2 - x^1), \dots, (x^k - x^1)$  are not linearly independent.  $\square$

A *polytope* is the convex hull of a non-empty finite set  $\{x^1, \dots, x^k\}$ . The polytope is a *simplex* if the points  $\{x^1, \dots, x^k\}$  are affinely independent. The set  $\{x \in \mathbb{R}^n \mid a^\top x = \beta\}$  where  $a$  is an  $m$  component nonzero vector is a *hyperplane* and the set  $\{x \in \mathbb{R}^n \mid a^\top x \leq \beta\}$  is a *closed half-space*. A hyperplane is an  $n - 1$  dimensional affine subspace of  $\mathbb{R}^n$ .

A *polyhedron* is the intersection of a finite number of closed half-spaces. That is, every polyhedron has the form  $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$  where  $A$  is an  $m \times n$  matrix of real numbers and  $b$  is an  $m$  component vector of real numbers. An inequality constraint  $(a^i)^\top x \geq b_i$  is *binding* for a given solution if it holds as an equality. We use the terms binding constraint and *tight constraint* synonymously throughout the text. If an inequality constraint does not hold as an equality for a given solution it has positive *slack*.

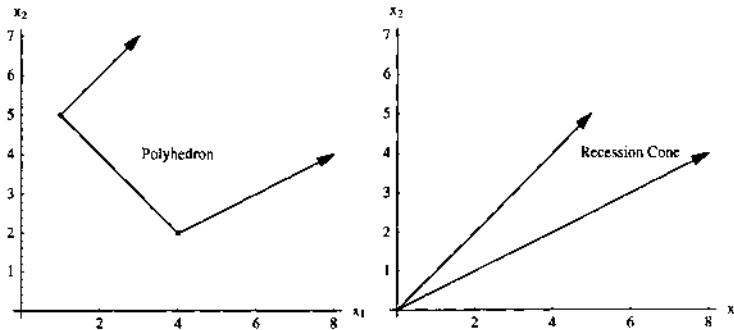
A set  $C \subseteq \mathbb{R}^n$  is a *cone* if and only if for all  $\bar{x} \in C$ ,  $\lambda \bar{x} \in C$  for all nonnegative  $\lambda$ . A *polyhedral cone* is defined by  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$ . A polyhedral cone is a cone, but not all cones are polyhedral cones. The *lineality space* of the polyhedral cone  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$  is  $\text{lin}(C) = \{x \in \mathbb{R}^n \mid Ax = 0\}$ . The polyhedral cone  $C$  is *pointed* if its lineality space has dimension 0. The *recession cone* of the polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  is  $\text{rec}(P) = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$ . The polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  is a *pointed polyhedron* if the recession cone of  $P$  is pointed. The recession cone of a polyhedron is illustrated in Figure A.2.

Let  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ . The point  $r \in \mathbb{R}^n$  is a *ray* of  $P$  if and only if for any  $x \in P$ , the set  $\{y \in \mathbb{R}^n \mid y = x + \lambda r, \lambda \in \mathbb{R}_+^1\} \subseteq P$ . A ray  $r$  of  $P$  is an *extreme ray* if there do not exist rays  $r^1, r^2, r^1 \neq \lambda r^2$  for any  $\lambda \in \mathbb{R}_+^1$  such that  $r = \frac{1}{2}r^1 + \frac{1}{2}r^2$ . The point  $\bar{x} \in P$  is an *extreme point* of  $P$  if and only if it cannot be written as a convex combination of other points in  $P$  distinct from  $\bar{x}$ . An alternative definition of an extreme point is given in Section A.3. One of the most elegant results in polyhedral theory is that for any polyhedron  $P$ , there is a finite set of points  $x^1, \dots, x^r$  such that

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}).$$

If  $P$  is a pointed polyhedron, then, without loss, the  $x^1, \dots, x^q$  are extreme points of  $P$  and the  $x^{q+1}, \dots, x^r$  are the extreme rays of  $\text{rec}(P)$ . This is an example of a *finite basis theorem*. Finite basis theorems are used extensively

Figure A.2 The Recession Cone of a Polyhedron



throughout the book. A traditional development of these results for polyhedra is given in Section A.4. A more unified treatment of important polyhedral results is given in Chapters 2–4.

Let  $P = \{(x, y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \mid Ax + By \geq d\}$ . The *projection* of the polyhedron  $P$  into the subspace of the  $y$  variables is

$$\text{proj}_y(P) := \{y \in \mathbb{R}^{n_2} \mid \text{there exists } x \in \mathbb{R}^{n_1} \text{ such that } (x, y) \in P\}.$$

Assume the rows of  $A$  which define  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  are  $a^i$ ,  $i = 1, \dots, m$ . Following Nemhauser and Wolsey [350], define  $I^- := \{i \mid a^i x = b_i, \text{ all } x \in P\}$  and  $I^\geq := \{1, \dots, m\} \setminus I^-$ . Let  $A^-$  denote the submatrix of  $A$  indexed by  $I^-$ . Similarly for  $A^\geq$ . If  $x \in P$  and  $a^i x > b_i$  for all  $i \in I^\geq$  then  $x$  is an *inner point* of  $P$ . If  $I^- = \emptyset$  then every inner point is an *interior point*.

**Proposition A.2 (Rank)** *If the polyhedron  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\} \neq \emptyset$ , then  $\dim(P) + \text{rank}(A^-) = n$ .*

The proof of Proposition A.2 is left as Exercise A.6.

**Proposition A.3 (Carathéodory)** *For any subset  $M$  of  $\mathbb{R}^n$  with  $\dim(\text{aff}(M)) = n$ , the convex hull  $\text{conv}(M)$  is the set of all convex combinations of at most  $n + 1$  points from  $M$ .*

**Proof:** Let  $y \in \text{conv}(M)$  and suppose at least  $r > n + 1$  points are required to express  $y$ , i.e.

$$y = \sum_{k=1}^r \lambda_k x^k, \quad \lambda_k > 0 \quad \text{and} \quad \sum_{k=1}^r \lambda_k = 1.$$

Since  $\dim(\text{aff}(M)) = n$ , it follows that there can be at most  $n + 1$  affinely independent points in  $M$ . Then, without loss,

$$x^1 = \sum_{k=2}^r \theta_k x^k, \quad \sum_{k=2}^r \theta_k = 1.$$

Since  $x^1 = \sum_{k=2}^r \theta_k x^k$  and  $y = \sum_{k=1}^r \lambda_k x^k$ , it follows that

$$\begin{aligned} y &= \sum_{k=1}^r \lambda_k x^k + \beta(x^1 - \sum_{k=2}^r \theta_k x^k) \\ &= (\lambda_1 + \beta)x^1 + \sum_{k=2}^r (\lambda_k - \beta\theta_k)x^k. \end{aligned}$$

for all real  $\beta$ . Since all  $\lambda_k > 0$  and at least one  $\theta_k$  is positive, there is a  $\beta > 0$  such that  $\lambda_k - \beta\theta_k \geq 0$  for  $k = 2, \dots, r$  and at least one  $(\lambda_k - \beta\theta_k)$  is zero. Since

$$(\lambda_1 + \beta) + \sum_{k=2}^r (\lambda_k - \beta\theta_k) = \beta + \sum_{k=1}^r \lambda_k - \beta \sum_{k=2}^r \theta_k = \sum_{k=1}^r \lambda_k = 1,$$

$y$  is a convex combination of at most  $r - 1$  points in  $M$ . This logic can be repeated until  $r = n + 1$  and the proposition is proved.  $\square$

In the Carathéodory proposition we *are not* saying that every  $y$  is a convex combination of the *same* set of points so this result is not as strong as the finite basis theorem for polyhedrons. Of course,  $M$  is an arbitrary subset of  $\mathbb{R}^n$ . Give an example of a convex set which does not have a finite basis.

**Theorem A.4 (Separating Hyperplane Theorem)** *Let  $C$  be a closed convex set in  $\mathbb{R}^n$  and  $y \notin C$ . Then there exists a hyperplane strictly separating  $y$  from  $C$ .*

### A.3 FACES OF POLYHEDRA

Let  $P \subseteq \mathbb{R}^n$  be a polyhedron. The set  $F \subseteq P$  is a *face* of  $P$  if there exists  $c \in \mathbb{R}^n$ ,  $c_0 \in \mathbb{R}^1$  such that  $F = \{x \in P \mid c^\top x = c_0\}$  and for all  $x \in P$ ,  $c^\top x \geq c_0$ .

If  $F$  is a face of the polyhedron  $P$  and  $F \neq \emptyset, F \neq P$ , then  $F$  is a *proper face* of  $P$ . Throughout, we use the notation  $Ax \geq b$  and  $(a^i)^\top x \geq b_i, i = 1, \dots, m$  interchangeably, depending on which is more convenient.

**Proposition A.5** *If  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\} \neq \emptyset$ , then  $F \neq \emptyset$  is a face of  $P$  if and only if there is a subsystem of  $Ax \geq b$  indexed by an  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\}$ .*

**Proof:** Assume there is an index set  $I$  and  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\} \neq \emptyset$ . Show  $F$  is a face of  $P$ . Define  $c := \sum_{i \in I} a^i$  and  $c_0 := \sum_{i \in I} b_i$ . Define  $F' = \{x \in P \mid c^\top x = c_0\}$ . Show  $F' = F$ . Since  $c^\top x = c_0$  is the sum of the constraints that define  $F$  it follows that  $F \subseteq F'$ . Show  $F' \subseteq F$ . Let  $\bar{x} \in F'$ . Then  $\bar{x} \in P$  which implies  $(a^i)^\top \bar{x} \geq b_i, i \in I$ . By definition of  $c, c_0, \sum_{i \in I} (a^i)^\top \bar{x} = \sum_{i \in I} b_i$ . Then  $(a^i)^\top \bar{x} = b_i$  for all  $i \in I$  and  $F' = F$ . The fact that  $c^\top x \geq c_0$  for all  $x \in P$  follows from

$$c^\top x = (\sum_{i \in I} a^i)^\top x \geq (\sum_{i \in I} b_i) = c_0.$$

Thus  $F$  is a face of  $P$ .

Now assume  $F = \{x \in P \mid c^\top x = c_0\} \neq \emptyset$  is a face of  $P$ . Show there exists  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\}$ . First observe that if  $\bar{x} \in F$  then at least one constraint in the system  $A\bar{x} \geq b$  is tight. Suppose not, i.e.  $A\bar{x} > b$ . Then define  $\hat{x}$  by  $\hat{x}_i = \bar{x}_i + \epsilon$  if  $c_i < 0$  and  $\hat{x}_i = \bar{x}_i - \epsilon$  if  $c_i > 0$ . Then for sufficiently small  $\epsilon$ ,  $A\hat{x} \geq b$ ,  $c^\top \hat{x} < c_0$  and we have a contradiction. Then for  $\bar{x} \in F$  define  $I(\bar{x}) := \{i \mid (a^i)^\top \bar{x} = b_i\}$ . We have shown that  $I(\bar{x})$  cannot be the null set for every  $\bar{x} \in F$ .

Next observe that for any  $\bar{x} \in F$ ,  $\min\{c^\top x \mid x \in P\} = \min\{c^\top x \mid (a^i)^\top x = b_i, i \in I(\bar{x})\} = c_0$  since  $\bar{x}$  is an optimal solution to the first minimization problem, and any constraints with positive slack can be deleted from the problem leaving only the tight constraints  $i \in I(\bar{x})$ . Since  $\{x \mid (a^i)^\top x = b_i, i \in I(\bar{x})\}$  is an affine set, and the minimization of a linear function exists over this set, it follows that all points in the set are optimal, i.e for all points in the affine set  $c^\top x = c_0$ . Then for any  $x \in P$  such that  $(a^i)^\top x = b_i, i \in I(\bar{x})$ ,  $c^\top x = c_0$ .

Define the family of sets  $\mathcal{G}(F) := \{I \mid I = I(\bar{x}) \text{ for some } \bar{x} \in F\}$ . Since the family  $\mathcal{G}(F)$  is finite there exists a finite set of points  $x^1, \dots, x^q$  in  $F$  such that  $\mathcal{G}(F) = \{I(x^i) \mid i = 1, \dots, q\}$ . Next define the index set  $G$  by  $G := \cap_{i=1}^q I(x^i)$ . Show  $G \neq \emptyset$ . The point  $\bar{x} = \sum_{i=1}^q (1/q)x^i$  is in  $F$  since it is a convex combination

of points in  $F$ . We have shown above that  $I(\bar{x}) \neq \emptyset$ . By definition of  $I(\bar{x})$ ,  $h \in I(\bar{x})$  implies  $h \in I(x^i)$ ,  $i = 1, \dots, q$ . Thus,  $I(\bar{x}) \subseteq G$  and  $G$  is not empty.

Define  $F' = \{x \in P \mid (a^i)^\top x = b_i, i \in G\}$ . Show  $F' = F$ . Let  $\hat{x} \in F$ . Then there exists  $h \in \{1, \dots, q\}$  such that  $I(\hat{x}) = I(x^h)$ . Then  $\hat{x} \in F'$  since  $G \subseteq I(\hat{x})$ . Let  $\hat{x} \in F'$ . We need only show that  $c^\top \hat{x} = c_0$ . This was shown earlier where we proved for any  $x \in P$  such that  $(a^i)^\top x = b_i$ ,  $i \in I(\bar{x})$ ,  $c^\top x = c_0$ .  $\square$

**Alternate Proof With Duality:** Now assume  $F = \{x \in P \mid c^\top x = c_0\}$  is a face of  $P$ . Show there exists  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\}$ . This is easily done through linear programming duality. The reader not familiar with linear programming duality should skip this proof. Duality is introduced in Section 2.6 of Chapter 2. Consider the linear program  $\min\{c^\top x \mid x \in P\}$ . If  $F = \{x \in P \mid c^\top x = c_0\} \neq \emptyset$  there is an optimal solution  $\bar{x}$  to this linear program with optimal solution value  $c_0$ . Let  $I$  index the constraints of  $Ax \geq b$  with strictly positive values for the optimal dual variables  $\bar{u}$ . By strong duality  $c = \sum_{i \in I} \bar{u}_i a^i$ ,  $c_0 = \sum_{i \in I} \bar{u}_i b_i$  and the result follows immediately.  $\square$

A zero dimensional face of  $P$  is an *extreme point* of  $P$ . The face  $F$  is a *facet* of  $P$  if  $\dim(F) = \dim(P) - 1$ . When working with more general convex sets it is necessary to define a face in a different manner, i.e. not as a supporting hyperplane.

**Proposition A.6** *Let  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  be a nonempty polyhedron and let  $a^i$  for  $i = 1, \dots, m$  denote the  $i$ th row of matrix  $A$ . If  $F$  is a proper face of  $P$  then the following conditions are equivalent:*

1. *there is an  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \mid (a^i)^\top x = b_i, i \in I\}$ ;*
2.  *$F$  is an affine subspace;*
3.  *$F$  does not contain any proper subsets which are proper faces of  $P$ ;*
4. *there is an  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\} = \{x \in P \mid (a^i)^\top x = b_i, i \in I, (a^i)^\top x > b_i, i \notin I\}$ ;*
5. *there is an  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\}$  and  $\text{rank}(A) = \text{rank}(A')$  where  $A'$  is the matrix consisting of the rows from  $A$  indexed by  $I$ .*

**Proof:** Show condition 1 implies condition 2. Let  $x^1, x^2 \in F$  and  $\lambda_1, \lambda_2$  be arbitrary real numbers such that  $\lambda_1 + \lambda_2 = 1$ . Then  $(a^i)^\top(\lambda_1 x^1 + \lambda_2 x^2) = \lambda_1(a^i)^\top x^1 + \lambda_2(a^i)^\top x^2 = \lambda_1 b_i + \lambda_2 b_i = b_i$  for all  $i \in I$  and the result follows.

Show condition 2 implies condition 3. Prove the contrapositive. Since  $F$  is a proper face, by Proposition A.5 there exists  $I \subseteq \{1, \dots, m\}$  such that  $F = \{x \in P \mid a^i x = b_i, i \in I\}$ . If  $F$  contains a proper subset which is also a proper face then there exists  $x^1, x^2 \in F$  and  $q \notin I$  such that  $(a^q)^\top x^1 = b_q$  and  $(a^q)^\top x^2 > b_q$ . Take the affine combination of  $x^1, x^2$  with  $\lambda_1 = 2$  and  $\lambda_2 = -1$ . Then  $(a^q)^\top(2x^1 - x^2) < b_q$  so  $F$  cannot be an affine subspace.

Show condition 3 implies condition 4. Trivial.

Show condition 4 implies condition 5. Prove the contrapositive by showing  $\text{rank}(A') < \text{rank}(A)$  implies not 4. If  $\text{rank}(A') < \text{rank}(A)$  there is an index set  $J \subseteq \{1, \dots, m\}$  such that  $I \subset J$  and  $\text{rank}(A'') = \text{rank}(A) = |J|$  where  $A''$  denotes the submatrix of  $A$  formed by taking the rows indexed by  $J$ . Then there is a solution  $x^1$  to  $A''x = b''$ . It is not necessarily true that  $x^1 \in P$ . However, there is an  $x^2 \in F$  so it is possible to find an  $x^3 = \lambda x^1 + (1 - \lambda)x^2$  with  $\lambda \in (0, 1]$  such that  $x^3 \in F$  but  $a^i x^3 = b^i$  for at least one  $i \in J \setminus I$ . This implies not 4.

Show condition 5 implies condition 1. Prove the contrapositive. Assume there is an index set  $I$  and face  $F = \{x \in P \mid (a^i)^\top x = b_i, i \in I\}$ . Again, assume  $I$  indexes every constraint such that  $(a^i)^\top x = b_i$  for all  $x \in P$ . Let  $F' = \{x \mid (a^i)^\top x = b_i, i \in I\}$ . If  $F' \neq F$ , then there exists  $x^1 \in F$  and  $x^2 \in F'$  such that  $x^2 \notin P$ . Then there exists  $q \in \{1, \dots, m\} \setminus I$  such that  $(a^q)^\top x^2 < b_q$ . But  $(a^i)^\top x^2 = b_i$  for all  $i \in I$  and  $(a^q)^\top x^1 \geq b_q$ . Then  $a^q$  cannot be a linear combination of the  $a^i$ ,  $i \in I$ . If it were, then

$$(a^q)^\top x^2 = \sum_{i \in I} \lambda_i (a^i)^\top x^2 = \sum_{i \in I} \lambda_i b_i = \sum_{i \in I} \lambda_i (a^i)^\top x^1 = (a^q)^\top x^1 \geq b_q,$$

but  $(a^q)^\top x^2 < b_q$ . Therefore  $\text{rank}(A') < \text{rank}(A)$ .  $\square$

A proper face  $F$  which satisfies any of the equivalent conditions 1-5 is a *minimal face*.

**Corollary A.7** If  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  is not empty, then every minimal face  $F$  of  $P$ , has  $\dim(F) = n - \text{rank}(A)$ .

**Proof:** By condition 1 of Proposition A.6, if  $F$  is a minimal face of  $P$  there is an index set  $I$  such that  $F = \{x \mid (a^i)^\top x = b_i, i \in I\}$ . Let  $A'$  index the submatrix

of  $A$  indexed by  $I$ . By Proposition A.2 the dimension of  $F$  is  $\dim(F) = n - \text{rank}(A')$ . But  $\text{rank}(A') = \text{rank}(A)$  for every minimal face by condition 5 of Proposition A.6. Thus, if  $F$  is a minimal face,  $\dim(F) = n - \text{rank}(A)$ .

**Corollary A.8** *If  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$  is not empty and  $\text{rank}(A) = n$ , then  $F$  is a minimal face of  $P$  if and only if  $F$  is an extreme point of  $P$ .*

Corollary A.8 is very significant for linear programming. The simplex algorithm for linear programming works by moving from extreme point to extreme point of the underlying polyhedron. This is justified by the following theorem.

**Theorem A.9 (Fundamental Theorem of Linear Programming)** *If the linear program  $\min \{c^\top x \mid Ax = b, x \geq 0\}$  has an optimal solution, then there is an optimal extreme point solution to this linear program. If there is not an optimal solution, then the linear program is either unbounded or infeasible.*

Extreme points are often defined in terms of basic feasible solutions. If  $A$  has rank  $m$ , there is an ordered index set  $B$ , of cardinality  $m$ , such that the columns of  $A$  indexed by  $B$  are linearly independent. The columns indexed by  $B$  are a *basis* and the variables  $x_i, i \in B$  are called *basic variables*. By  $x_B$  we mean an  $m$  component vector of variables with components  $x_{B_i}$  for  $i = 1, \dots, m$ . Given the system,  $Ax = b$ ,  $x \geq 0$ , the solution  $\bar{x}_B = A_B^{-1}b$ ,  $\bar{x}_N = 0$  is a *basic feasible solution* if  $\bar{x}_B$  is nonnegative.

**Proposition A.10 (Basic Feasible Solution)** *The vector  $\bar{x}$  is a basic feasible solution of the system  $Ax = b$ ,  $x \geq 0$  if and only if  $\bar{x}$  is an extreme point of  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ .*

The next proposition says that the highest dimensional faces are all that we need to describe a polyhedron. A constraint  $(a^q)^\top x \geq b_q$  which is part of the system  $(a^i)^\top x \geq b_i$  is a *redundant constraint* if and only if removing the constraint from the system does not alter the feasible region of the associated polyhedron.

**Proposition A.11** *If  $P = \{x \in \mathbb{R}^n \mid (a^i)^\top x \geq b_i, i = 1, \dots, m\}$  and  $q \in I^{\geq}$ , then  $F = \{x \in P \mid (a^q)^\top x = b_q\}$  is a facet of  $P$  if and only if  $(a^q)^\top x \geq b_q$  is not a redundant constraint.*

**Proof:** Assume  $(a^q)^\top x \geq b_q$  where  $q \in I^{\geq}$ , is not a redundant constraint. Then  $F = \{x \in P \mid (a^q)^\top x = b_q\}$  is a proper face of  $P$  and we need only show that it has rank  $\dim(P) - 1$ . Do this by showing that the rank of the equality set for  $F$  is equal to  $\text{rank}(A^=) + 1$ . It suffices to show that  $a^q$  is a not a linear combination of the rows of  $A^=$ . Let  $\hat{b} = \sum_{i \in I^=} \lambda_i b^i$  for any set of  $\lambda_i$ . If  $a^q = \sum_{i \in I^=} \lambda_i a^i$ , then  $q \in I^{\geq}$  implies  $b_q < \hat{b}$ . But this would also imply  $(a^q)^\top x \geq b_q$  is redundant since  $(a^q)^\top x \geq \hat{b}$  is both a tighter and redundant constraint. Therefore,  $a^q$  is not a linear combination of rows from  $A^=$  so the rank of the equality set of  $F$  is at least one greater than the rank of  $A^=$ . It is left as Exercise A.25 to show that the rank of  $F$  does not exceed the rank of  $A^=$  by more than one. By Proposition (A.2),

$$\dim(F) = n - (\text{rank}(A^=) + 1) = \dim(P) - 1$$

and  $F$  is a facet of  $P$ .

Next show that if  $F$  is a facet of  $P$  then  $(a^q)^\top x \geq b_q$  is not redundant. If  $F$  is a facet, then the rank of the equality set of  $F$  is equal to the rank of the equality set of  $P$  plus one. Then there exists a point  $x^1$ , such that  $(a^i)^\top x^1 > b_i$  for all  $i \in I^{\geq} \setminus \{q\}$  and  $(a^q)^\top x^1 = b_q$ . Let  $x^* = x^1 + \epsilon(x^1 - x^2)$  where  $x^2$  is an arbitrary inner point of  $P$ . (Note we are taking an affine combination of  $x^1, x^2$ .) For sufficiently small  $\epsilon$  we have  $(a^q)^\top x^* < b_q$  and  $(a^i)^\top x^* \geq b_i$ , for all  $i \neq q$ . Therefore  $(a^q)^\top x \geq b_q$  is not redundant.  $\square$

## A.4 FINITE BASIS THEOREMS

First consider polyhedral cones. Let  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$ . Assume matrix  $A$  has  $m$  rows and define for  $I \subseteq \{1, \dots, m\}$ ,

$$C(I) := \{x \in C \mid (a^i)^\top x = 0, i \in I\}.$$

Define  $C(\emptyset) = C$ . Next define a family of subsets  $\mathcal{F}$ , on the ground set  $\{1, \dots, m\}$  by

$$\mathcal{F} := \{I \subset \{1, \dots, m\} \mid \text{there is an } \bar{x} \in C(I), (a^i)^\top \bar{x} > 0 \text{ for all } i \notin I\}.$$

By definition of  $C(I)$  and  $I^=$ ,  $C = C(I^=)$ . If  $Ax > 0$  has a solution then  $I^= = \emptyset$  and  $\emptyset \in \mathcal{F}$ . If  $I^= \subset I$  and  $I \in \mathcal{F}$  then  $C(I)$  is a proper face of  $C$ . Why? The following lemma is used in the proof of our main result.

**Lemma A.12** If  $\bar{x}$  is an inner point of  $C$  and  $z^1$  is an element of  $C$ , but  $z^1$  is not in  $\text{lin}(C)$ , then there exists a  $\bar{\lambda} > 0$  and an index set  $I \subseteq \{1, \dots, m\}$  such that  $\bar{x} - \bar{\lambda}z^1 \in C(I)$ , and  $\dim(C(I)) < \dim(C)$ .

**Proof:** Since  $(a^i)^\top \bar{x} > 0$  and for all  $i \in I^2$ , and  $(a^i)^\top z^1 > 0$  for at least one  $i \in I^2$ , there exists a sufficiently small  $\bar{\lambda} > 0$  and index set  $I$  such that

$$\begin{aligned} (a^i)^\top \bar{x} - (a^i)^\top (\bar{\lambda}z^1) &= 0, \text{ for } i \in I, \\ (a^i)^\top \bar{x} - (a^i)^\top (\bar{\lambda}z^1) &> 0, \text{ for } i \notin I. \end{aligned}$$

Thus  $\bar{x} - \bar{\lambda}z^1 \in C(I)$  by definition of  $C(I)$ . By construction  $I^2 \subset I$  so  $C(I)$  is a proper face of  $C$  which means  $\dim(C(I)) < \dim(C)$ .  $\square$

The set  $I \in \mathcal{F}$  is *maximal with respect to containment* if  $I \subset J$  implies  $J \notin \mathcal{F}$ . For cones, the minimal face is always the lineality space which is not too interesting. Therefore, we introduce the concept of a minimal proper face for cones. If  $I \in \mathcal{F}$  and  $I$  is maximal with respect to containment, then  $C(I)$  is a *minimal proper face* of  $C$ .

**Lemma A.13** If  $I \in \mathcal{F}$  is maximal with respect to containment, then  $\dim(C(I)) = \dim(\text{lin}(C)) + 1$ .

**Proof:** If  $I \in \mathcal{F}$  is maximal with respect to containment, then for any  $k \in \{1, \dots, m\} \setminus I$ ,  $C(I \cup \{k\}) = \text{lin}(C)$ . By definition of  $\mathcal{F}$ , if  $k \in \{1, \dots, m\} \setminus I$  there is an  $\bar{x} \in C(I)$  such that  $(a^k)^\top \bar{x} > 0$  and  $(a^i)^\top \bar{x} = 0$ , for all  $i \in I$ . Thus, the rank of the matrix with rows  $a^k$  and  $a^i, i \in I$  exceeds the rank of the matrix with rows  $a^i, i \in I$  by exactly one and the result follows.  $\square$

**Lemma A.14** If  $I \in \mathcal{F}$ , then the following conditions are equivalent:

1.  $\dim(\text{lin}(C)) = 0$  and  $I$  is maximal with respect to containment;
2.  $\text{rank}(A) = n$  and  $\dim(C(I)) = 1$ ;
3. for any nonzero  $\bar{x} \in C(I)$ ,  $C(I) = \{x \mid x = \lambda \bar{x}, \lambda \geq 0\}$ ;
4. for any nonzero  $\bar{x} \in C(I)$ ,  $\bar{x}$  is an extreme ray of  $C$ .

**Proof:** Show condition 1 implies condition 2. This follows directly from Lemma A.13.

Show condition 2 implies condition 3. Since  $0 \in C(I)$  and  $\dim(C(I)) = 1$ , there exists a nonzero  $\bar{x} \in C(I)$  such that any point in  $C(I)$  is an affine combination of  $0$  and  $\bar{x}$ . That is, if  $\hat{x} \in C(I)$ ,  $\hat{x} = \lambda\bar{x} + (1 - \lambda)0$ , i.e.  $\hat{x} = \lambda\bar{x}$ . Since  $\dim(\text{lin}(C)) = 0$ , without loss take  $\lambda$  to be nonnegative.

Show condition 3 implies condition 4. Let  $x^1, x^2$  be any elements of  $C$  such that  $\bar{x} = (1/2)x^1 + (1/2)x^2$ . Then  $(a^i)^\top x^1 = 0$  and  $(a^i)^\top x^2 = 0$  for all  $i \in I$ . Thus,  $x^1, x^2 \in C(I)$  and are nonnegative multiples of each other. Therefore,  $\bar{x}$  is an extreme ray of  $C$ .

Show condition 4 implies condition 1. Prove the contrapositive and assume that either  $I$  is not maximal with respect to containment or the lineality space has dimension greater than 0. Then there exists  $J \subseteq \{1, \dots, m\}$  (if  $I$  not maximal, then  $J \in \mathcal{F}$ , if the lineality space has dimension greater than 0 take  $J$  to be  $\{1, \dots, m\}$ ) such that  $I \subset J$  and there is a  $k \in J \setminus I$  such that  $(a^k)^\top \bar{x} > 0$ , and there is also a nonzero  $\bar{y} \in C(J)$ . Then for sufficiently small  $\epsilon > 0$  the points  $\bar{x} + \epsilon\bar{y}$  and  $\bar{x} - \epsilon\bar{y}$  are in  $C$ . But  $(a^k)^\top \bar{x} > 0$  and  $(a^k)^\top \bar{y} = 0$  imply  $\bar{x} + \epsilon\bar{y}$  is not a positive scalar multiple of  $\bar{x} - \epsilon\bar{y}$ . Therefore,  $\bar{x}$  not an extreme ray of  $C$ .  $\square$

The following classic Theorem is due to Minkowski [335]. It is proved in Chapter 3 using inverse projection. It plays a central role throughout this book.

**Theorem A.15 (Minkowski)** *Let  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$ . Then there exists  $x^1, \dots, x^t \in \mathbb{R}^n$  such that  $C = \text{cone}(\{x^1, \dots, x^t\})$  where each  $x^i$  is either an element of the lineality space of  $C$ , or is an element of a minimal proper face of  $C$ .*

**Proof:** For each set  $I \in \mathcal{F}$  indexing a minimal proper face of  $C$  select an arbitrary  $z^h \in C(I)$  such that  $z^h$  is an inner point of  $C(I)$ . Assume there are  $z^1, \dots, z^s$  such points. Since  $\text{lin}(C)$  is a linear space of finite dimension it is generated by  $r$  linearly independent points  $y^i$ ,  $i = 1, \dots, r$ . Show

$$C = \text{cone}(\{z^1, \dots, z^s, y^1, \dots, y^r, -y^1, \dots, -y^r\}).$$

It suffices to show  $C \subseteq \text{cone}(\{z^1, \dots, z^s, y^1, \dots, y^r, -y^1, \dots, -y^r\})$  since the reverse direction is trivial. Let  $\bar{x} \in C$  and show  $\bar{x}$  can be written as a conic combination of the candidate points.

Proof by induction on the dimension of  $C$ . Assume the dimension of  $C$  is 0. Then  $C$  is a single point which is the origin. This is also the lineality space

and is generated by the  $y^i$ . In this case there is only one  $y^i$  and it is the zero vector.

Assume the result is true for  $\dim(C) \leq n$  and prove this implies the result for  $\dim(C) = n + 1$ . If  $\bar{x} \in \text{lin}(C)$  the result is immediate so assume  $\bar{x}$  is not in the lineality space of  $C$ . If  $\bar{x}$  is not in the lineality space of  $C$  then  $\bar{x}$  is either an inner point or it is not. If  $\bar{x}$  is not an inner point then  $\bar{x} \in C(I)$  for some  $I \in \mathcal{F}$  and  $C(I)$  is a proper face of  $C$ . Then the dimension of  $C(I)$  is  $n$  or less. Since  $\text{lin}(C) = \text{lin}(C(I))$  any minimal proper face of  $C(I)$  is also a minimal proper faces of  $C$ . Therefore, an inner point for every minimal proper face of  $C(I)$  can be selected from the  $z^1, \dots, z^s$ . The result now follows by the induction hypothesis. Assume  $\bar{x}$  is an inner point of  $C$ . Apply Lemma A.12 to the points  $\bar{x}$  and  $z^1$ . Then  $\bar{x} - \bar{\lambda}z^1 \in C(I)$  with  $\dim(C(I)) \leq n$ . Again, if a face is a minimal proper face of  $C(I)$  then it is also a minimal proper face of  $C$ . Therefore, by the induction hypothesis

$$\bar{x} - \bar{\lambda}z^1 = \sum_{i=1}^s \bar{\alpha}_i z^i + \sum_{i=1}^r \bar{\beta}_i y^i + \sum_{i=1}^r \bar{\gamma}_i (-y^i)$$

for  $\bar{\alpha}_i, \bar{\beta}_i, \bar{\gamma}_i \geq 0$ . Then,

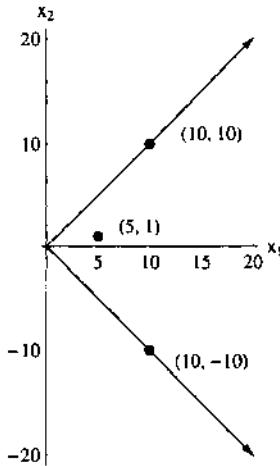
$$\bar{x} = \bar{\lambda}z^1 + \sum_{i=1}^s \bar{\alpha}_i z^i + \sum_{i=1}^r \bar{\beta}_i y^i + \sum_{i=1}^r \bar{\gamma}_i (-y^i)$$

and the proposition is proved.  $\square$

**Example A.16** Consider the cone  $C = \{(x_1, x_2) | x_1 - x_2 \geq 0, x_1 + x_2 \geq 0\}$ . Then  $\mathcal{F} = \{\{1\}, \{2\}, \emptyset\}$ . Pick as the candidate points  $z^1 = (10, 10)$  for the point from minimal proper face  $C(\{1\}) = \{(x_1, x_2) \in C | x_1 - x_2 = 0\}$  and  $z^2 = (10, -10)$  for the point from the minimal face  $C(\{2\}) = \{(x_1, x_2) \in C | x_1 + x_2 = 0\}$ . Since the dimension of the lineality space is zero we don't need any points from the lineality space. Let  $\bar{x} = (5, 1)$ . The cone  $C$ , the generators, and the point  $(5, 1)$  are illustrated in Figure A.3. This point is an inner point, and for small  $\lambda > 0$ ,

$$\begin{aligned} (a^1)^\top \bar{x} - (a^1)^\top \lambda z^1 &= 4 - \lambda(0) > 0 \\ (a^2)^\top \bar{x} - (a^2)^\top \lambda z^1 &= 6 - \lambda(20) > 0 \end{aligned}$$

Take  $\bar{\lambda} = 3/10$ . Then  $(5, 1) - \bar{\lambda}(10, 10) = (2, -2)$  which is in face  $C(\{2\})$ . The selected point in this face is  $z^2 = (10, -10)$ . Thus,  $(5, 1) - (3/10)(10, 10) = (1/5)(10, -10)$  so  $(5, 1) = (3/10)(10, 10) + (1/5)(10, -10)$ .

**Figure A.3** A Cone and Generators

**Example A.17** Consider the cone,  $C = \{(x_1, x_2) | x_1 + x_2 \geq 0\}$ . Then  $\mathcal{F} = \{\emptyset\}$ . Pick as the candidate points  $(7, 0)$  for point in the minimal proper face  $C(\emptyset)$  and  $(1, -1)$ ,  $(-1, 1)$  for points in the lineality space. Let  $\bar{x} = (5, 5)$  be the selected point to write as a conic combination of the basis. This point is not in the lineality space so find  $\lambda > 0$  such that  $a^1\bar{x} - \lambda a^1(7, 0) = 0$ , that is  $10 - \lambda 7 = 0$ . Then  $\bar{\lambda} = 10/7$  and the projected point is

$$(5, 5) - (10/7)(7, 0) = (-5, 5).$$

Then  $(-5, 5) = 5(-1, 1)$  and  $(5, 5) = (10/7)(7, 0) + 5(-1, 1)$ . In this example  $C(\emptyset) = C$  and  $C$  is a minimal proper face but not a proper face. Why?

An interesting property of polyhedral cones is that when  $\text{rank}(A) = n$ , the unique minimal face is the origin. When  $\text{rank}(A) < n$  there are no extreme points or extreme rays (See Lemmas A.13 and A.14) and the unique minimal face is the lineality space. When  $\text{rank}(A) = n$  the minimal proper faces are the extreme rays of the cone. These are enough to generate the cone.

**Corollary A.18** If  $C = \{x \in \mathbb{R}^n \mid Ax \geq 0\}$  and  $\text{rank}(A) = n$ , then there exist  $x^1, \dots, x^q$  extreme rays of  $C$  such that  $C = \text{cone}\{x^1, \dots, x^q\}$ .

**Theorem A.19 (Finite Basis Theorem)** *If  $P = \{x \in \mathbb{R}^n \mid Ax \geq b\} \neq \emptyset$ , then there exists  $x^1, \dots, x^r$  in  $P$  such that*

$$P = \text{conv}(\{x^1, \dots, x^q\}) + \text{cone}(\{x^{q+1}, \dots, x^r\}).$$

**Proof:** Define the cone  $C = \{(x, x_0) \mid Ax - x_0 b \geq 0, x_0 \geq 0\}$  in  $\mathbb{R}^{n+1}$ . By Minkowski's theorem, there exists a set of points in  $\mathbb{R}^{n+1}$ ,  $(x^1, x_0^1), \dots, (x^r, x_0^r)$  such that  $C = \text{cone}(\{(x^1, x_0^1), \dots, (x^r, x_0^r)\})$ . Without loss, assume that  $x_0^i > 0$  implies  $x_0^i = 1$  and that  $x_0^i = 1$  for  $i = 1, \dots, q$  and  $x_0^i = 0$  for  $i = q+1, \dots, r$ . That is, if the component corresponding to  $x_0$  is positive, then it is equal to 1.

Let  $\bar{x} \in P$ . Then  $(\bar{x}, 1) \in C$ . Then there are nonnegative  $\bar{\lambda}_1, \dots, \bar{\lambda}_r$  such that

$$(\bar{x}, 1) = \sum_{i=1}^q \bar{\lambda}_i (x^i, 1) + \sum_{i=q+1}^r \bar{\lambda}_i (x^i, 0).$$

Then

$$\bar{x} = \sum_{i=1}^q \bar{\lambda}_i x^i + \sum_{i=q+1}^r \bar{\lambda}_i x^i.$$

where  $\sum_{i=1}^q \bar{\lambda}_i = 1$  and we have the desired result. Note that set inclusion in the reverse direction is trivial.  $\square$

**Corollary A.20** *A bounded polyhedron is a polytope.*

**Corollary A.21** *A polytope is finitely generated by the extreme points of the polytope.*

**Theorem A.22 (Weyl)** *If the cone  $C$  is finitely generated then it is a polyhedral cone.*

A proof of Weyl's theorem is given in Chapter 2. There is an analogous result for polytopes.

**Proposition A.23** *If  $P$  is a polytope, then it is a polyhedron.*

## A.5 INNER PRODUCTS, SUBSPACES AND ORTHOGONAL SUBSPACES

The vector space used in this text is the one defined by vector addition and scalar multiplication in  $\mathbb{R}^n$ . If

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \text{then} \quad x + y := \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix}$$

and the *inner product* or *dot product* is

$$x^\top y := x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

The *Euclidean norm* is  $\|x\| := \sqrt{x^\top x}$ . In terms of geometry, by the law of cosines,

$$\|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2\|x\|\|y\|\cos(\theta)$$

and by definition of inner product,

$$\begin{aligned} \|x - y\|^2 &= (x - y)^\top (x - y) \\ &= x^\top x + x^\top (-y) + (-y)^\top x + (-y)^\top (-y) \\ &= \|x\|^2 + \|y\|^2 - 2x^\top y. \end{aligned}$$

Therefore,

$$\|x\|^2 + \|y\|^2 - 2\|x\|\|y\|\cos(\theta) = \|x\|^2 + \|y\|^2 - 2x^\top y,$$

and

$$x^\top y = \|x\|\|y\|\cos(\theta),$$

where  $\theta$  is the angle between the vectors  $x, y$ .

If  $H \subseteq \mathbb{R}^n$  is a linear space under vector addition and scalar multiplication, then  $H$  is a *subspace* of  $\mathbb{R}^n$ .

**Lemma A.24** *If  $H$  is a subspace of  $\mathbb{R}^n$  and  $H^\perp := \{x \in \mathbb{R}^n \mid x^\top y = 0, \text{ for all } y \in H\}$ , then  $H^\perp$  is a subspace of  $\mathbb{R}^n$ .*

The subspace  $H^\perp$  defined in Lemma A.24 is the *orthogonal subspace* of  $H$ .

**Lemma A.25** *If  $H$  is a subspace of  $\mathbb{R}^n$ , then  $\dim(H) + \dim(H^\perp) = n$  and  $H = (H^\perp)^\perp$ .*

**Proof:** Let  $x^1, \dots, x^{q_1}$  be an orthogonal basis for  $H$  and  $y^1, \dots, y^{q_2}$  and orthogonal basis for  $H^\perp$ . Clearly  $q_1 + q_2 \leq n$ . Assume  $q_1 + q_2 < n$ . Then there exists a vector  $z$  that is not a linear combination of  $x^1, \dots, x^{q_1}$  and  $y^1, \dots, y^{q_2}$ . Then by Gramm-Schmidt there is a  $\bar{z}$  orthogonal to  $x^1, \dots, x^{q_1}$ , but not linearly dependent on  $y^1, \dots, y^{q_2}$ . But,  $\bar{z} \in H^\perp$  and this contradicts the fact that  $y^1, \dots, y^{q_2}$  define a basis for  $H^\perp$ . The second part of the proof is trivial.  $\square$

**Lemma A.26** *If  $H$  is a subspace of  $\mathbb{R}^n$ , then there exists an  $n \times q$  matrix  $B$  such that  $H = \{x \in \mathbb{R}^n \mid x = By, y \in \mathbb{R}^q\}$ .*

**Proof:** If  $H$  is a subspace of  $\mathbb{R}^n$ , then there are vectors  $b^1, \dots, b^q$  in  $\mathbb{R}^n$  which are a basis for  $H$  and any point in  $H$  can be written as a linear combination of the  $b^i$  vectors. Let  $b^i$ ,  $i = 1, \dots, q$  define the  $q$  columns of  $B$ .  $\square$

**Lemma A.27** *If  $H \subseteq \mathbb{R}^n$ , then  $H$  is a subspace if and only if there exists a  $q \times n$  matrix  $A$  such that  $H = \{x \in \mathbb{R}^n \mid Ax = 0\}$  and  $H^\perp = \{x \in \mathbb{R}^n \mid x = A^\top y, y \in \mathbb{R}^q\}$ .*

**Proof:** Assume  $H$  is a subspace of  $\mathbb{R}^n$ . Then  $H^\perp$  is a subspace of  $\mathbb{R}^n$  by Lemma A.24. Let  $a^1, \dots, a^q$  be set of linearly independent vectors that span  $H^\perp$ . Define  $A^\top$  to be the  $n \times q$  matrix with columns  $a^1, \dots, a^q$ . The result is now immediate from Lemmas A.25 and A.26. The other direction of this Lemma is immediate.  $\square$

Let  $x$  be a point in  $\mathbb{R}^n$  and  $H$  a subspace of  $\mathbb{R}^n$ . The point  $y \in H$  is the *projection of  $x$  into  $H$*  if and only if  $(x - y) \in H^\perp$ . This projection is unique.

**Lemma A.28** *If  $x \in \mathbb{R}^n$  and  $H$  is a subspace of  $\mathbb{R}^n$ , then there are unique  $y \in H$  and  $z \in H^\perp$  such that  $x = y + z$ .*

**Proposition A.29 (Linear Projection)** *Let  $H = \{x \in \mathbb{R}^n \mid Ax = 0\}$ , where  $A$  is a  $q \times n$  matrix of rank  $q$  and let  $x \in \mathbb{R}^n$ . The projection of  $x$  into  $H$  is*

$$y = [I - A^\top (AA^\top)^{-1}A]x.$$

**Proof:** By Lemma A.28,  $x = y + z$  where  $y \in H$  and  $z \in H^\perp$ . Then by Lemma A.27 there exists  $w \in \mathbb{R}^q$  such that  $x = y + A^\top w$ . Since  $y \in H$ ,  $Ay = 0$  and solving for  $y$  gives

$$0 = Ay = Ax - (AA^\top)w,$$

which implies,

$$w = (AA^\top)^{-1}Ax.$$

Then,  $x = y + A^\top(AA^\top)^{-1}Ax$  and

$$y = [I - A^\top(AA^\top)^{-1}A]x.$$

The matrix  $[I - A^\top(AA^\top)^{-1}A]$  is called a *projection matrix*.

**Example A.30** Let  $H = \{(x_1, x_2) | x_1 - x_2 = 0\}$  and  $x = (1, 2)$ . In this example  $A = [1, -1]$ . The projection matrix is the 2 by 2 matrix with each element equal to 1/2. The projection into  $H$  is the point  $(1.5, 1.5)$ .

## A.6 EXERCISES

A.1 Prove that if  $M \subseteq \mathbb{R}^n$ , then  $\text{aff}(M)$  is a closed set.

A.2 Let  $x^1, x^2$  be distinct points in  $\mathbb{R}^2$ . Show that the set of points defined by

$$\{(x_1, x_2) | (x_1, x_2) = \lambda_1(x_1^1, x_2^1) + \lambda_2(x_1^2, x_2^2), \lambda_1 + \lambda_2 = 1\}$$

determine a straight line in  $\mathbb{R}^2$ .

A.3 Carefully explain why  $\text{rank}(A) = n$  is required in the hypothesis of Corollary A.18.

A.4 If  $\Gamma \subseteq \mathbb{R}^n$ , the *polar*  $\Gamma^*$  of  $\Gamma$  is defined by the set

$$\Gamma^* = \{z \in \mathbb{R}^n | z^\top x \leq 1, x \in \Gamma\}.$$

If  $\Gamma$  is a polyhedron containing the origin prove the following:

- a)  $\Gamma^*$  is a polyhedron;
- b)  $\Gamma^{**} = \Gamma$ . (Where  $\Gamma^{**} = \{z \in \mathbb{R}^n | z^\top x \leq 1, x \in \Gamma^*\}$ ).

A.5 If  $C \subseteq \mathbb{R}^n$  is a closed set, then  $\text{conv}(C)$  is a closed set. Prove this or give a counter-example.

A.6 Prove Proposition A.2.

A.7 Assume that  $P \subseteq \mathbb{R}^n$  is a polytope and that  $T : \mathbb{R}^n \rightarrow \mathbb{R}^m$  is a linear transformation from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ . Prove that  $T(P)$  is a polytope in  $\mathbb{R}^m$ .

A.8 A point  $\bar{x}$  is an extreme point of the convex set  $C$  if and only if there do not exist distinct  $x^1, x^2 \in C$  such that  $\bar{x}$  is a convex combination of  $x^1, x^2$ . Prove that this definition is equivalent to the definition of an extreme point as a zero dimensional face. You may assume  $C$  is a polyhedron.

A.9 Prove Proposition A.10

A.10 Let  $x^1, x^2$  be distinct feasible solutions to the system  $Ax = b$ ,  $x \geq 0$  and define  $J^1 := \{j | x_j^1 > 0\}$  and  $J^2 := \{j | x_j^2 > 0\}$ . If  $J^2 \subset J^1$  and  $x^2$  is an extreme point of the system then  $x^1$  is not an extreme point of the system. Prove this conjecture or give a counter example.

A.11 If there is an optimal solution to  $\min\{c^\top x | Ax = b\}$ , then every  $x$  such that  $Ax = b$  is optimal.

A.12 Let  $V$  be the vector space of continuous real-valued functions defined on  $[0, 1]$ , and let

$$(f, g) = \int_0^1 f(t)g(t)dt.$$

Prove that  $(\cdot, \cdot)$  defines an inner product over  $V$ .

A.13 Prove Lemma A.24.

A.14 Prove Lemma A.28.

A.15 Show that an affine space cannot have any proper faces.

A.16 Consider the following system of equations.

$$\begin{aligned}(1/2)x_1 + x_2 &\geq 4 \\ x_1 - x_2 &\geq -6 \\ x_2 &\geq 2 \\ x_1 &\geq 0\end{aligned}$$

- Graph the recession cone for the polyhedron described above.
- Graph the lineality space of the polyhedron defined above.
- Find the extreme rays of the recession cone for the polyhedron described above which will generate the cone.

A.17 The convex hull of the points  $(2, 2)$ ,  $(4, 4)$ ,  $(6, 0)$  and  $(3, -5)$  is a polytope (by definition). A polytope is also a polyhedron. We defined a polyhedron to be the intersection of a finite number of half-spaces.

- a. Give the *equations* of the half-spaces that define this polytope.
- b. Show that the point  $(2, 2)$  can be written as an affine combination of  $(4, 4)$ ,  $(6, 0)$  and  $(3, -5)$ .

A.18 Give the coordinates of all the extreme points of the polyhedron defined by the constraints:

$$\begin{aligned} 3x_1 + x_2 + 2x_3 + x_4 + x_5 &= 1 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

Justify why the points selected are really extreme points.

A.19 Characterize the extreme points of the polyhedron defined by

$$\begin{aligned} \sum_{i=1}^n a_i x_i &= b \\ x_i &\geq 0, \quad i = 1, \dots, n \end{aligned}$$

A.20 Prove the fundamental theorem of linear programming.

A.21 Define the polyhedral cone  $C$  by the constraints:

$$\begin{aligned} x_1 - 4x_2 + 3x_3 &\leq 0 \\ x_1 - x_2 + x_3 &\leq 0 \\ -x_2 &\leq 0 \\ -x_3 &\leq 0 \end{aligned}$$

- a. Find  $\text{lin}(C)$ .
- b. Find an inner point in every minimal proper face of  $C$ .
- c. Find a finite basis for  $C$  consisting of inner points from the minimal proper faces of  $C$  and write the point  $(0, 10, 1)$  as a conic combination of points from this chosen basis.

A.22 Prove Corollary A.18.

A.23 Prove that if  $P$  is a polyhedron in  $\mathbb{R}^n$  then  $P$  can be written as  $P = L + H$  where  $L$  is a linear space and  $H$  is a pointed polyhedron contained in  $L^\perp$ .

A.24 Prove that if the affine space  $\{x \in \mathbb{R}^n | Ax = b\}$  is not empty, then the maximum number of affinely independent solutions of  $Ax = b$  is  $n + 1 - \text{rank}(A)$ .

A.25 If  $P = \{x \in \mathbb{R}^n | (a^i)^\top x \geq b_i, i = 1, \dots, m\}$  and  $q \in I^{\geq}, (a^q)^\top x \geq b_q$  is not a redundant constraint of  $P$ , then the rank of the equality set of  $F = \{x \in P | a^q x = b_q\}$  cannot exceed  $A^= + 1$ .

# B

---

## COMPLEXITY THEORY

### B.1 INTRODUCTION

There are many algorithms presented in this book and it is important to be precise in measuring the complexity of an algorithm. One way to evaluate the complexity of an algorithm is to count the number of additions, subtractions, multiplications, divisions and comparisons required in the worst case. Although average case, or how the algorithm behaves on real data is important, in this book all of the complexity analysis is based on a worst case running time. For example, if  $A$  is an  $n \times n$  matrix, Gaussian elimination applied to the system  $Ax = b$  requires approximately  $n^3/3$  additions and multiplications and  $n^2/2$  divisions. In general, if  $A$  is an  $n \times n$  matrix, Gaussian elimination has complexity  $O(n^3)$ . This big  $O$  notation is used throughout the text. If  $f$  and  $g$  are functions from  $\mathbb{Z}_+$  to  $\mathbb{R}$ , then  $f(n) = O(g(n))$  if there exists a constant  $k$ , and an integer  $n_0$  such that  $|f(n)| \leq k|g(n)|$  for all  $n \geq n_0$ . In the case of Gaussian elimination,  $f(n)$  is the total number of arithmetic operations required for an  $n \times n$  matrix and  $g(n) = n^3$ . The model of computation that counts the number of additions, subtractions, divisions, multiplications and comparisons as a function of the number of numbers input to the problem is called the *arithmetic model* of computation. In the arithmetic model, every operation takes one time unit regardless of the size of the number. There are problems with simply counting arithmetic operations when measuring complexity. The first problem has to do with how rational numbers are represented in a computer.

Rational numbers are those which can be expressed as  $p/q$  where  $p$  and  $q$  are integers and  $q$  is not 0. On a real computer, these numbers are stored in exponential form, not as a ratio of integers. They are stored as

$$s * M * B^{e-E}. \quad (\text{B.1})$$

In expression (B.1),  $s$  is the sign bit (a 0 for positive or 1 for negative),  $M$  is the mantissa and  $B$  is the radix and  $e - E$  the exponent. In the exponent term,  $E$  is a fixed integer called the bias. Generally,  $B = 2$ , although on the IBM 360/370 the radix is 16. In a typical 32 bit word, the sign bit is a single bit, the exponent  $e$  is stored as an eight bit positive integer, the mantissa is 23 bits, and  $E$  is set equal to 127. Since  $e$  is a positive integer, the exponent  $e - E$  can be either positive or zero. The bit pattern in the mantissa is normalized by making the leftmost bit a 1. By normalizing in this way a phantom bit is created since it is not necessary to store the one.

**Example B.1** Consider the following 32 bit representation.

1 01111110 10100000000000000000000000000000

In this example,  $s = 1$ ,  $e = 126$ , and the mantissa is 1.101 so the number is

$$-1 * 1.101 * 2^{126-127} = -1 * 1.625 * 2^{-1} = .8125$$

Even a seemingly innocuous rational number like  $0.1 = 1/10$  cannot be stored exactly using 32 bits (or any finite number of bits using a binary scheme). An alternative to this storage scheme is to actually carry out the arithmetic in infinite precision and store the rational number as the ratio of the integers  $p$  and  $q$ . However, this brings about a second problem, namely what if  $p$  and  $q$  get “too big” and exceed the maximum size of an integer that can be stored in one word. Even if an algorithm only generates integers, storing those integers may still represent a problem. Consider the following simple algorithm.

### Algorithm B.2 (Squaring a Real Number)

```

for i = 1, ..., n + 1
    read x[i]
end
for i = 1, ..., n
    x[i + 1] ← x[i] * x[i]
end

```

If we simply count the number of inputs to the algorithm, this algorithm has complexity  $O(n)$  since there are  $n$  multiplications. But what if  $x[1] = 2$  and the computer word length is 32 bits. It is left as an exercise to show that

upon termination of Algorithm B.2 that  $x[n+1] = 2^{2^n}$ . With a word length of 32, and  $n > 5$ ,  $x[n+1]$  can no longer be represented using only one word. Therefore, saying this algorithm is  $O(n)$  is not correct if the size of the words used in the computation is considered and infinite precision is required. A real computer would require time  $2^n$  to write the answer so it is not reasonable to assume that an arithmetic operation takes one unit of time regardless of the size of the number. Next, consider the Euclidean algorithm for finding the gcd of the integers  $p$  and  $q$ . See (4.1)-(4.2) in Chapter 4. There are two inputs to the algorithm, namely  $p$  and  $q$ , yet the number of arithmetic operations is  $O(\log_2(\max\{|p|, |q|\}))$ . Thus, the Euclidean algorithm is polynomial as a function of the number of bits required to encode the input. However, it is not polynomial in the arithmetic model since the complexity of the algorithm is not a polynomial function of the number of inputs which is two. Clearly when talking about the complexity of an algorithm both the number of inputs and the size of the encoding must be considered. The binary encoding of integers, rational numbers, matrices of rational numbers, and solution sizes is the topic of Section B.2.

An algorithm is *polynomial* if: 1) the number of arithmetic operations is bounded by a polynomial function of the number of inputs and the size of the inputs, and 2) the space required is a polynomial function of the size of the inputs. The algorithm is *strongly polynomial* if: 1) the number of arithmetic operations is bounded by a polynomial function of the number of inputs, and 2) the space required to do the computations is a polynomial function of the size of the encoding. The Euclidean algorithm is a polynomial algorithm but is not a strongly polynomial algorithm.

A model of computation that considers both the number of inputs and their size is the Turing model. This model is formally defined in Section B.3. In Section B.4 important complexity classes based on the Turing model are described. In Section B.5 the concept of Satisfiability is introduced. Satisfiability is used in Section B.6 to define the important class of  $\mathcal{NP}$ -complete problems. For a detailed presentation of this material see Garey and Johnson [166]. In Section B.7 a strongly polynomial, infinite precision Gaussian elimination algorithm is given. That is, rational numbers are stored as the ratio of two integers and we show during pivoting that the integers grow as a polynomial function of the size of the input. Once it is demonstrated that Gaussian elimination is strongly polynomial we no longer worry about infinite precision or word size. All of the polynomial algorithms in the text can be made to run in polynomial time in infinite precision using the methods of Section B.7. Exercises are provided in Section B.8.

## B.2 SOLUTION SIZES

If  $n$  is an integer, the *size* of  $n$  is

$$\langle n \rangle := 1 + \lceil \log_2(|n| + 1) \rceil$$

bits. The size of a rational number  $r = p/q$  where  $p$  and  $q$  are coprime integers is

$$\langle r \rangle := \langle p \rangle + \langle q \rangle.$$

The size  $\langle b \rangle$  of a rational vector  $b$ , is the sum of the sizes of its components, the size  $\langle A \rangle$  of a rational matrix  $A$ , is the sum of the sizes of its elements. The size of solutions to a linear is important throughout the text. First, it is necessary to bound the size of a determinant of a rational matrix.

**Lemma B.3 (Hadamard Inequality)** *If  $A$  is an  $n \times n$  matrix then  $|\det(A)| \leq \prod_{i=1}^n \|a^i\|$  where  $a^i, i = 1, \dots, n$  is column  $i$  of  $A$ .*

The following lemma is used throughout the book. See Grötschel, Lovász, and Schrijver [212] for a proof.

**Lemma B.4** *If  $A$  is an  $n \times n$  rational matrix, then  $|\det(A)| \leq 2^{\langle A \rangle - n^2} - 1$ .*

Now consider the linear program in standard form.

$$(LP) \quad \begin{aligned} & \min c^\top x \\ & \text{s.t. } Ax = b \\ & \quad x \geq 0 \end{aligned}$$

If the input data  $c, A, b$  to the linear program  $(LP)$  are integer, then the size of linear program  $(LP)$  is

$$\begin{aligned} L &:= \sum_{i=1}^m \sum_{j=1}^n (1 + \lceil \log_2(1 + |a_{ij}|) \rceil) + \sum_{j=1}^m (1 + \lceil \log_2(1 + |b_j|) \rceil) \\ &\quad + \sum_{j=1}^n (1 + \lceil \log_2(1 + |c_j|) \rceil). \end{aligned} \tag{B.2}$$

The size of the constraints is

$$L' := \sum_{i=1}^m \sum_{j=1}^n (1 + \lceil \log_2(1 + |a_{ij}|) \rceil) + \sum_{j=1}^m (1 + \lceil \log_2(1 + |b_j|) \rceil). \tag{B.3}$$

**Lemma B.5** *If the input data  $c, A, b$  to the linear program (LP) are integer, and if  $\bar{x}$  is a basic feasible solution of (LP) then for  $\bar{x}_j > 0$ ,*

$$\frac{1}{2^{L'-m^2} - 1} \leq |\bar{x}_j| \leq 2^{L'-m^2} - 1.$$

**Lemma B.6** *If the input data  $c, A, b$  to the linear program (LP) are integer, and if  $\bar{x}$  is a basic feasible solution of (LP), then  $|c^\top \bar{x}| < 2^L$ .*

**Proof:**

$$\begin{aligned} |c^\top \bar{x}| &= \left| \sum_{j=1}^n c_j \bar{x}_j \right| \\ &\leq \sum_{j=1}^n |c_j \bar{x}_j| \\ &= \sum_{j=1}^n |c_j| |\bar{x}_j| \\ &\leq \max_{j=1, \dots, m} \{|c_j|\} \left( \sum_{j=1}^n |\bar{x}_j| \right) \end{aligned}$$

By hypothesis  $\bar{x}$  is a basic feasible solution so at most  $m$  components of  $\bar{x}$  are positive. Since  $(m/2^{m^2}) < 1$  for  $m \geq 1$ , by Lemma B.5,

$$\sum_{j=1}^n |\bar{x}_j| < 2^{L'} - m < 2^{L'}.$$

But  $|c_j| \leq 2^{\langle c \rangle}$ ,  $j = 1, \dots, n$  so

$$|c^\top \bar{x}| < \max\{|c_j| \mid j = 1, \dots, m\} 2^{L'} \leq 2^{\langle c \rangle} 2^{L'} = 2^L. \quad \square$$

### B.3 THE TURING MACHINE

The period from around 1920-1940 was truly a Golden Age for mathematical logic. During this time logicians were trying to prove completeness of axiom systems and mechanize theorem proving. They were not entirely successful in

their attempts, but one significant result is an abstract conceptualization of computing – the *Turing machine* named after the great British logician Alan Turing. The Turing machine is a very general model of computation, however Turing did show that some problems are so hard that no Turing machine algorithm can be given for solving them.

Let  $\Omega$  denote a finite set which is called the *alphabet*. Often,  $\Omega = \{0, 1\}$ . Let  $\Omega^*$  denote the set of all strings of symbols from  $\Omega$ . Thus the string  $z = 11001001$  is an element of  $\Omega^*$ . The number of elements of the alphabet  $\Omega$  in  $z$  determine the *size* of  $z$  again denoted by  $(z)$ . The  $z$  in the previous example has size 8. A language  $\mathcal{L}$ , is a subset of strings of the alphabet, i.e.  $\mathcal{L} \subseteq \Omega^*$ . The *decision problem* or *language recognition problem* corresponding to  $\mathcal{L}$  is given  $z \in \Omega^*$  determine if  $z \in \mathcal{L}$ .

**Example B.7** Consider the decision problem “given an  $m \times n$  integer matrix  $A$ , decide if there is a nonzero solution to  $Ax \leq 0$ .” How is this problem stated using the formalism developed so far? Let  $\Omega = \{0, 1, -, \dagger, \ddagger\}$ . The 0, 1 symbols are used to encode each integer of  $A$  in binary,  $-$  denotes a negative number, the  $\dagger$  symbol denotes the end of a coefficient, the  $\ddagger$  symbol denotes the end of a row. Consider the matrix

$$\begin{bmatrix} 0 & -5 \\ -2 & 7 \end{bmatrix}.$$

Then the appropriate  $z \in \Omega^*$  is  $z = 0\dagger-101\dagger\dagger-10\dagger111\dagger\dagger$ . Is  $z \in \mathcal{L}$ ?

We could just as easily use a decimal encoding of the data and let

$$\Omega = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, \dagger, \ddagger\}.$$

In this case  $z = 0\dagger-5\dagger\dagger-2\dagger7\dagger\dagger$ . The binary encoding is more common.

A single tape *Turing machine* can be thought of as a tape of infinite length divided into an infinite number of tape cells or squares. There is a tape head which is positioned over exactly one tape square at each point of time. This tape head can read and write symbols in the tape alphabet  $\Omega'$ . The tape alphabet is defined by  $\Omega' = \Omega \cup \{\#\}$  where  $\#$  is the “blank” symbol. At time zero, every square in the Turing machine not corresponding to the input string has the blank symbol in it. Think of the tape head as being controlled by a finite state control “black box”. Depending upon the current state of the system and the symbol in the square under the tape head, the finite state control box will dictate: 1) a move of the tape head either one square to the right, +1, one square to the left, -1 or remain over the current square, 2) which new symbol

is to be written over the tape square after the move, and 3) a new state for the system. Let  $Q$  denote the possible set of states. Then a Turing machine is completely defined by the next move function

$$\delta : Q \times \Omega' \rightarrow Q \times \Omega' \times \{-1, 0, +1\}.$$

Think of the next move function  $\delta$  as actually being the Turing machine. The *time complexity*,  $T(n)$  of a Turing machine  $\delta$  is the maximum number moves made in processing an input of length  $n$ , taken over all possible inputs of length  $n$ . The *space complexity*,  $S(n)$  is the maximum distance (from the initial tape square at time 0) of the tape head moves, from the initial tape square, to the right or to the left in processing any input of length  $n$ . Polynomial time complexity implies polynomial space complexity.

## B.4 COMPLEXITY CLASSES

### B.4.1 The Class $\mathcal{P}$ .

The class  $\mathcal{P}$  is the class of all languages  $\mathcal{L}$  accepted by deterministic Turing machines of polynomial time complexity. That is,  $\mathcal{L} \subseteq \Omega^*$  is in  $\mathcal{P}$  if and only if there is a polynomial time complexity Turing machine  $\delta$ , such that the string  $z$  is accepted by  $\delta$  if and only if  $z \in \mathcal{L}$ . That is,  $\mathcal{L}$  is exactly the set of strings accepted by  $\delta$ .

### *Encoding the Data*

Turing machines do not directly solve decision problems such as the one given in Example B.7. The Turing machine solves an *encoding* of the problem into a language  $\mathcal{L}$ . In Example B.7 two encodings (binary and decimal) were given which led to distinct languages defined on different alphabets. It is possible that one encoding of a decision problem results in a language in  $\mathcal{L}_1$  which is accepted by a Turing machine of polynomial time complexity and another encoding results in a language  $\mathcal{L}_2$  for which there is no known Turing machine of polynomial time complexity.

Consider the unary encoding of the data, i.e. an integer  $n$  uses  $|n|$  bits. Then the length of the encoding grows polynomially with the coefficient magnitude. However, if we choose binary encoding then the growth of the encoding is only logarithmic. Since polynomial growth is so much faster than logarithmic an

algorithm polynomial in unary input might well be exponential in logarithmic input. For example, assume that in unary encoding the time complexity is  $O(n)$ . Since

$$O(\text{number bits unary encoding}) = O(2^{(n)})$$

this algorithm is exponential in the binary encoding of the data. To say that a decision problem is polynomially solvable, we must use the binary (or similar) encoding of the data, not the unary. Actually the base does not matter as long as it is not unary since  $\log_a(b) = (\log_a(|n|)/\log_b(|n|))$ .

### *Problem Reduction*

It is often the case that being able to find a solution to one problem enables one to find a solution to another problem through a simple transformation or subroutine. Loosely speaking, we might say problem one “reduces” to problem two if a polynomial algorithm for problem two implies the existence of a polynomial algorithm for problem one. More formally, there is a *polynomial transformation* from the language  $\mathcal{L}_1 \subseteq \Omega_1^*$  to the language  $\mathcal{L}_2 \subseteq \Omega_2^*$  if and only if there is a function  $\psi : \Omega_1^* \rightarrow \Omega_2^*$  such that

1. there is a Turing machine of polynomial time complexity which can evaluate  $\psi(z)$ ;
2.  $z \in \Omega_1^*$  is in  $\mathcal{L}_1$  if and only if  $\psi(z) \in \mathcal{L}_2$ .

If there is a polynomial transformation from  $\mathcal{L}_1$  to  $\mathcal{L}_2$  we write  $\mathcal{L}_1 \propto \mathcal{L}_2$  (read  $\mathcal{L}_1$  reduces to  $\mathcal{L}_2$ ). The reduction given here is known as *Karp reducibility*. One might also say that decision problem one reduces to decision problem two if there is an algorithm for solving problem one, which uses an algorithm for problem two as a subroutine, and runs in polynomial time assuming each call of the subroutine takes unit time. This is often called *polynomial or Turing reducibility*.

### B.4.2 The Class $\mathcal{NP}$

There are several ways to think of this class of problems. Here is the first. The class  $\mathcal{NP}$  is the set of all languages  $\mathcal{L} \subset \Omega^*$  such that

1. there exist a language  $\mathcal{L}' \subseteq \Omega^* \times \Omega^*$  which is in  $\mathcal{P}$ ;

2. there is a polynomial function  $\phi : \mathcal{L} \rightarrow \mathbb{R}_+^1$ ;
3.  $z \in \mathcal{L}$  if and only if there exists  $y \in \Omega^*$ ,  $(z, y) \in \mathcal{L}'$  and  $\langle y \rangle \leq \phi(\langle z \rangle)$ .

The string  $y$  in the definition above is often called a *certificate of proof*. How  $y$  is found is irrelevant, it is not necessary to be able to find  $y$  in an efficient fashion. Trivially,  $\mathcal{P} \subseteq \mathcal{NP}$  since it is valid to take  $\mathcal{L}' := \{(z, \emptyset) | z \in \mathcal{L}\}$ . Now for a nontrivial example of a problem in  $\mathcal{NP}$ .

**Example B.8 (0/1 Integer Programming)** *Given a rational  $m \times n$  matrix  $A$  and a rational  $m$  vector  $b$  does there exist a 0/1 solution to  $Ax \geq b$ . The language  $\mathcal{L}$  is the binary encoding of all  $(A, b)$  which have feasible 0/1 solutions. Then  $y$  can be any feasible 0/1 solution since the size of  $y$  is obviously bounded by a polynomial function of the encoding of  $(A, b)$  and there is Turing machine of polynomial time complexity which concludes for feasible  $y$  that  $((A, b), y) \in \mathcal{L}'$ .*

**Example B.9 (Integer Programming)** *Given a rational  $m \times n$  matrix  $A$  and a rational  $m$  vector  $b$ , does there exist an integer solution to  $Ax \geq b$ . This is similar to Example B.8 except that it is more complicated to show that if there is an integer solution to  $Ax \geq b$ , then there is a solution which is “small enough.” Indeed, there is a small enough solution. If there is an integer solution to  $Ax \geq b$  there is one of size at most  $6n^3\theta$  where  $\theta$  is the facet complexity of the polyhedron defined by  $Ax \leq b$ . The facet complexity is the maximum of  $n$  and the size of the system  $Ax \leq b$ . See Schrijver [403] and the references contained therein.*

The term  $\mathcal{NP}$  stands for *nondeterministic polynomial*. The word nondeterministic has the following motivation. Since  $\langle y \rangle \leq \phi(\langle z \rangle)$  we could find the required  $y$  in a nondeterministic fashion, i.e. by guessing. There are at most  $|\Omega|^{\phi(\langle z \rangle)}$  solutions. Thus, membership in  $\mathcal{NP}$  implies it is possible solve the yes/no decision problem by a process of total enumeration of a polynomial-length vector. This logic give rise to the idea of a *nondeterministic Turing machine*.

The deterministic Turing machine was defined by the next move function

$$\delta : Q \times \Omega' \rightarrow Q \times \Omega' \times \{-1, 0, +1\}.$$

The key word here is *function*, given a state and symbol the new move, state, and symbol are uniquely determined. With a nondeterministic Turing machine this is not the case. More than one move is allowed, i.e.

$$\delta : Q \times \Omega' \subseteq Q \times \Omega' \times \{-1, 0, +1\}.$$

A language  $\mathcal{L}$  is in  $\mathcal{NP}$  if it is accepted by a nondeterministic Turing machine with polynomial time complexity. The ability of the nondeterministic Turing machine to select a point from a subset of  $Q \times \Omega' \times \{-1, 0, +1\}$  corresponds to the guessing idea discussed above.

Given the ease of showing hard problems such as the integer linear programming are in  $\mathcal{NP}$ , one might get the idea that it is easy to show any decision problem is in  $\mathcal{NP}$ . This is not true.

**Example B.10 (Kth Heaviest Subset)** *Given the integers  $c_1, \dots, c_n, K, L$ , find  $K$  distinct subsets  $S_1, \dots, S_K$  such that*

$$\sum_{j \in S_i} c_j \geq L, \quad i = 1, \dots, K.$$

*What “certificate” is needed to answer yes to this problem? Apparently, a listing of  $K$  such subsets is required and  $K$  can be as large as  $2^n - 1$ .*

### B.4.3 The Class $co\text{-}\mathcal{NP}$

The complement of the language  $\mathcal{L} \subseteq \Omega^*$  is the language  $\Omega^* \setminus \mathcal{L}$ . If the complement of  $\mathcal{L}$  is in  $\mathcal{NP}$ , then  $\mathcal{L}$  is in  $co\text{-}\mathcal{NP}$ . The set  $co\text{-}\mathcal{NP}$  is the class of all such languages. Another way to say this is that  $\mathcal{L}$  is in  $co\text{-}\mathcal{NP}$  if and only if there is an  $\mathcal{L}' \subseteq \Omega^* \times \Omega^*$  which is in  $\mathcal{P}$ , a polynomial function  $\phi : \Omega^* \rightarrow \mathbb{R}_+^1$ , and

$$z \in \Omega^* \setminus \mathcal{L} \text{ if and only if there exists } y \in \Omega^*, (z, y) \in \mathcal{L}' \text{ and } \langle y \rangle \leq \phi(\langle z \rangle).$$

Determining that  $\mathcal{L} \in co\text{-}\mathcal{NP}$  is very difficult since given a certificate  $y$ , such that  $(z, y)$  has a no answer, provides very little information, i.e. we cannot conclude that  $z \notin \mathcal{L}$ . To illustrate consider the 0/1 integer programming problem of Example B.8. To conclude that the  $z$  corresponding to the binary encoding of  $(A, b)$  was not in  $\mathcal{L}$  we would have to enumerate every binary solution. But this is not acceptable, since to conclude  $z \notin \mathcal{L}$  we would have to find an  $\mathcal{L}'$  satisfying the above definition. Furthermore, there is no obvious nondeterministic Turing machine for solving this problem. Notice also that for this example we could actually conclude  $z \notin \mathcal{L}$  by enumerating an exponential number of “clues,” since the size of a clue is bounded by  $n$ . In general this may not be the case, because we could have  $\mathcal{L} \notin \mathcal{NP}$ , but  $\mathcal{L} \in co\text{-}\mathcal{NP}$ .

**Example B.11** Consider the decision problem “given a real  $m \times n$  matrix  $A$  and  $m$  vector  $b$  does there exist a solution to the system  $Ax \geq b$ ?” Since this

decision problem is in  $\mathcal{P}$  it is in  $\mathcal{NP}$ . Is it in  $\text{co-}\mathcal{NP}$ ? That is, given  $A, b$ , is the problem of determining that there is no solution to this system in  $\mathcal{NP}$ ? By Farkas lemma, the system  $Ax \geq b$  does not have a solution if there is a nonzero solution to  $A^T y = 0$ ,  $y^T b > 0$ . This problem is obviously in  $\mathcal{P}$  also.

A problem is well characterized if it belongs to  $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ . Linear programming is well characterized.

## B.5 SATISFIABILITY

Classical propositional logic consists of a set of Boolean variables  $x_i$  (or atomic propositions or literals or predicates) together with the connectives  $\neg$  (negation),  $\vee$  (disjunction)  $\wedge$  (conjunction) and  $\rightarrow$  (implication) which are used to form statements. We replace  $x_1 \rightarrow x_2$  with  $\neg x_1 \vee x_2$ . We are interested in clauses that are in *conjunctive normal form (CNF)*, i.e. each formula is a conjunction of a series of clauses each of which is a pure disjunction of atomic propositions or their negation. The *satisfiability problem* is to determine a truth setting for the atomic propositions which make a formula in CNF form true. Denote by  $\mathcal{L}_{SAT}$  the set of all clauses in conjunctive normal form which are satisfiable.

**Lemma B.12**  $\mathcal{L}_{SAT} \in \mathcal{NP}$ .

We show how to model a Turing machine as a satisfiability problem over a Horn clause knowledge base. A clause with at most one positive atomic proposition is a *Horn clause*. Consider a single tape Turing machine with finite alphabet  $\Omega' = \{0, 1, b\}$ , finite states  $Q$ , and next move function  $\delta : Q \times \Omega' \rightarrow Q \times \Omega' \times \{-1, 0, 1\}$ . To model a Turing machine which accepts language  $L$  as a satisfiability problem on a Horn clause knowledge base define the literals:

1.  $v(t, c, q, \alpha)$  is true if, before move  $t$ , and after move  $t - 1$ , the machine is in state  $q \in Q$  with symbol  $\alpha \in \Omega'$  in tape square  $c$  and the tape head is over square  $c$ .
2.  $w(t, c, \alpha)$  is true if, before move  $t$ , and after move  $t - 1$ , tape square  $c$  contains symbol  $\alpha$  and the tape head is not over square  $c$ .

The truth of a literal involving tape square  $c$  at move  $t$  is completely determined by the next move function and the contents of tape squares  $c - 1$ ,  $c$ , and  $c + 1$

**Table B.1** Horn Clause Knowledge Base for Deterministic Turing Machine

RULES	
$w(t, c, \alpha) :-$	$w(t - 1, c - 1, \alpha_{c-1}), w(t - 1, c, \alpha), w(t - 1, c + 1, \alpha_{c+1})$
$w(t, c, \alpha) :-$	$v(t - 1, c - 1, q', \alpha_{c-1}), w(t - 1, c, \alpha), w(t - 1, c + 1, \alpha_{c+1})$ for all $\delta(q', \alpha_{c-1}) = (*, *, -1)$ or $\delta(q', \alpha_{c-1}) = (*, *, 0)$
$w(t, c, \alpha) :-$	$w(t - 1, c - 1, \alpha_{c-1}), v(t - 1, c, q', \beta), w(t - 1, c + 1, \alpha_{c+1})$ for all $\delta(q', \beta) = (*, \alpha, -1)$ or $\delta(q', \beta) = (*, \alpha, 1)$
$w(t, c, \alpha) :-$	$w(t - 1, c - 1, \alpha_{c-1}), w(t - 1, c, \alpha), v(t - 1, c + 1, q', \alpha_{c+1})$ for all $\delta(q', \alpha_{c+1}) = (*, *, 0)$ or $\delta(q', \alpha_{c+1}) = (*, *, 1)$
$v(t, c, q, \alpha) :-$	$v(t - 1, c - 1, q', \alpha_{c-1}), w(t - 1, c, \alpha), w(t - 1, c + 1, \alpha_{c+1})$ for all $\delta(q', \alpha_{c-1}) = (q, *, 1)$
$v(t, c, q, \alpha) :-$	$w(t - 1, c - 1, \alpha_{c-1}), v(t - 1, c, q', \beta), w(t - 1, c + 1, \alpha_{c+1})$ for all $\delta(q', \beta) = (q, \alpha, 0)$
$v(t, c, q, \alpha) :-$	$w(t - 1, c - 1, \alpha_{c-1}), w(t - 1, c, \alpha), v(t - 1, c + 1, q', \alpha_{c+1})$ for all $\delta(q', \alpha_{c+1}) = (q, *, -1)$
FACTS	
$v(1, 1, q_0, 1) \leftrightarrow$	$z_1 = 1$
$v(1, 1, q_0, 0) \leftrightarrow$	$z_1 = 0$
$w(1, c, 1) \leftrightarrow$	$z_c = 1, c = 2, \dots, n$
$w(1, c, 0) \leftrightarrow$	$z_c = 0, c = 2, \dots, n$
$w(1, c, b)$	$c = n + 1, \dots, \nu(n)$

at move  $t - 1$ . See Table B.1 for a list of the necessary Horn clauses. Table B.1 and Proposition B.13 are taken from Jeroslow et al. [243]. Assume, without loss, that the input language  $L$  tested is a set of binary strings, the length of the input string  $z$ , is  $n$ , the input string begins in square 1, the time complexity is  $\phi(n)$ ,  $q_0 \in Q$  is the state of the machine at time zero,  $q_A \in Q$  is the accepting state where  $z \in L$ , the total number of tape squares required (both input and workspace) is  $\nu(n)$ , and the tape head is initially over square 1. Predicates which do not use tape squares 1 through  $\nu(n)$  are not included in the knowledge base.

**Proposition B.13** *Let  $T$  be a deterministic Turing machine for deciding if a string  $z$  is in language  $\mathcal{L}$ . Then  $z$  is accepted by the Turing machine if and only if a literal corresponding to the accepting state can be set true in the Horn clause knowledge base given by Table B.1.*

**Proof:** It suffices to show for the Horn clause knowledge base of Table B.1 that i) for each move  $t$  exactly one  $v(t, c, q, \alpha)$  is true, and ii) that for each tape square  $c$  and each move  $t$  at most one of the literals  $w(t, c, \alpha)$ ,  $v(t, c, q, \alpha)$  is

true. These two conditions require that the tape head be above a unique tape square at every move, the machine be in a unique state at every move, and each tape square contains a unique symbol at each move.

Proceed by induction. For  $t = 1$  the result is obvious since either  $v(1, 1, q_0, 1)$  is true and  $v(1, 1, q_0, 0)$  false when  $z_1 = 1$ , or  $v(1, 1, q_0, 1)$  is false and  $v(1, 1, q_0, 0)$  true when  $z_1 = 0$ . Similarly for the  $w$  literals.

Assume i) and ii) are true for moves 1 through  $t$ . First show part i) for move  $t + 1$ , i.e. distinct  $v(t + 1, c, q, \alpha)$  and  $v(t + 1, c', q', \alpha')$  cannot both be true. There are two subcases to consider: ia)  $c \neq c'$  or  $q \neq q'$ , and ib)  $c = c'$ ,  $q = q'$ , and  $\alpha \neq \alpha'$ . From Table 1 it follows that if  $v(t + 1, c, q, \alpha)$  is true there must be a corresponding  $v$  literal true at move  $t$ . Similarly for  $v(t + 1, c', q', \alpha')$ . But  $c \neq c'$  or  $q \neq q'$  and the fact that a deterministic Turing machine only allows unique next move functions implies that at least two distinct  $v$  literals must be true at move  $t$  which contradicts the induction hypothesis so ia) cannot happen.

Next consider ib) with  $c = c'$ ,  $q = q'$ , but  $\alpha \neq \alpha'$ . If the tape head is to the left or right of  $c$  before move  $t$  and only one  $v$  literal is true, then from Table B.1 both  $w(t - 1, c, \alpha)$  and  $w(t - 1, c, \alpha')$  are true which contradicts ii) of the induction hypothesis. If the tape head is above square  $c$  at the start of move  $t$  then due to the uniqueness of the next move function, at least one of the  $w$  literals for square  $c$  must also be true in order to account for both  $\alpha$  and  $\alpha'$  occurring in true literals, again contradicting part ii) of the induction hypothesis. The proof of part ii) for move  $t + 1$  is similar.  $\square$

## B.6 $\mathcal{NP}$ -COMPLETENESS

Many important, but difficult, problems are in the class  $\mathcal{NP}$ . What if a polynomial algorithm were found for the decision problem corresponding to  $\mathcal{L}_1 \in \mathcal{NP}$  and it could be shown that for every  $\mathcal{L}_2 \in \mathcal{NP}$  that  $\mathcal{L}_2 \propto \mathcal{L}_1$ . This would imply  $\mathcal{P} = \mathcal{NP}$ . A polynomial algorithm for such an  $\mathcal{L}_1$  would imply the existence of a polynomial algorithm for any  $\mathcal{L}_2 \in \mathcal{NP}$ ! Such decision problems are clearly important and have a special name. The language  $\mathcal{L}_1 \subseteq \Omega^*$  is  $\mathcal{NP}$ -complete if and only if

1.  $\mathcal{L}_1 \in \mathcal{NP}$ ;
2. if  $\mathcal{L}_2 \in \mathcal{NP}$  then  $\mathcal{L}_2 \propto \mathcal{L}_1$ .

A decision problem with an encoding into a language that satisfies condition 2, but is not known to satisfy condition 1, is called  $\text{NP}$ -hard. This is because they are least as “hard” as any problem in  $\text{NP}$ . The satisfiability problem is  $\text{NP}$ -complete. This is a very famous result due to Cook [97].

**Theorem B.14 (Cook)** *SAT is  $\text{NP}$ -complete.*

**Proof:** Let  $\mathcal{L}_{SAT}$  denote the language of all satisfiable strings of clauses in CNF form defined on alphabet  $\Omega_{SAT} = \{x_1, \dots, x_n, \vee, \wedge, \neg\}$ , and let  $\mathcal{L} \subseteq \Omega^*$  be any language in  $\text{NP}$ . By Lemma B.12  $\mathcal{L}_{SAT} \in \text{NP}$  and it suffices to show  $\mathcal{L} \propto \mathcal{L}_{SAT}$ .

By definition of  $\text{NP}$  there exists a deterministic Turing machine  $\delta'$  which accepts language  $\mathcal{L}' \subseteq \Omega^* \times \Omega^*$  and has polynomial time complexity  $T(n + \phi(n))$ . Let  $\psi : \Omega^* \rightarrow \Omega_{SAT}^*$  be the function which maps strings from  $\Omega^*$  to  $\Omega_{SAT}^*$  by defining the string in  $\Omega_{SAT}^*$  to be the clauses defined by Table B.1. and the “facts” in Table B.1 corresponding to the  $n$  input bits which define  $z$ . Since  $\delta'$  has polynomial time complexity  $T(n + \phi(n))$  this mapping is done in polynomial time (with the appropriate Turning machine.) It is *not necessary* to know how to construct the Turing machine  $\delta'$ . It is only necessary to take a given  $\delta'$ , a  $z \in \Omega^*$ , and construct  $\psi(z) \in \Omega_{SAT}^*$  in polynomial time. The result now follows from Proposition B.13.  $\square$

Theorem B.14 implies that if the satisfiability problem can be solved in polynomial time, any problem in  $\text{NP}$  can be solved in polynomial time. The question of whether or not  $\mathcal{P} = \text{NP}$  is one of the most perplexing problems in computer science. Most people believe that the equality does not hold. If you can answer the question one way or the other, you will have an immediate dissertation. In fact, answering this question would be good enough to get you a chaired professorship in any U.S. university.

## B.7 COMPLEXITY OF GAUSSIAN ELIMINATION

Finding a solution to the system  $Ax = b$  where  $A$  is an  $m \times n$  matrix requires calculating the  $LU$  decomposition of  $PAQ$ . This involves  $O(\min\{m, n\}mn)$  arithmetic operations. Therefore, in order to prove that Gaussian elimination is strongly polynomial it is necessary to show that the numbers do not get “too

big." This means that in the Turing machine model of computation storing the numbers using their binary encoding requires only a polynomial amount of space as a function of the size of the input  $A, b$ . We are assuming that an infinite precision solution to  $Ax = b$  is required. If the word size is fixed and rounding is allowed, it is trivial to show that Gaussian elimination requires at most  $O(\min\{m, n\}mn)$  arithmetic operations. The problem in infinite precision is how to store the elements of the  $A^{(k)}$  because division is required in (13.1) and (13.4). One option is to write  $(p_1/q_1) + (p_2/q_2)$  as  $(p_1q_2 + p_2q_1)/q_1q_2$  and store the numerator and denominator as the integers  $(p_1q_2 + p_2q_1)$  and  $q_1q_2$ . Let's investigate this idea in more detail. In the strategies below we assume that  $A$  and  $b$  have integer components. If not, they can be converted to have integer components that have size which is a polynomial function of the size of the elements in  $A$  and  $b$ . Most of the discussion in this section is taken from Grötschel, Lovász, and Schrijver [212].

**Strategy 1:** Store any rational number  $p/q$  as integers  $p$  and  $q$  and do addition as described as above. In matrix  $A^{(k)}$  each element  $a_{ij}^{(k)}$  is written as  $f_{ij}^{(k)}/g^{(k)}$  where  $g^{(k)}$  is a common divisor of the elements in  $A^{(k)}$ . Since  $A$  is assumed to be integer take  $g^{(1)} = 1$  and  $f_{ij}^1 = a_{ij}^1 = a_{ij}$ . Then (13.1) is

$$\begin{aligned} a_{ij}^{(k+1)} &= \frac{f_{ij}^{(k)}}{g^{(k)}} - \frac{f_{ik}^{(k)}/g^{(k)}}{f_{kk}^{(k)}/g^{(k)}}(f_{kj}^{(k)}/g^{(k)}) \\ &= \frac{f_{ij}^{(k)}}{g^{(k)}} - \frac{f_{ik}^{(k)} f_{kj}^{(k)}}{f_{kk}^{(k)} g^{(k)}} \\ &= \frac{f_{ij}^{(k)} f_{kk}^{(k)} - f_{ik}^{(k)} f_{kj}^{(k)}}{f_{kk}^{(k)} g^{(k)}} \\ &= \frac{f_{ij}^{(k+1)}}{g^{(k+1)}} \end{aligned}$$

The integers  $f_{ij}^{(k+1)} := f_{ij}^{(k)} f_{kk}^{(k)} - f_{ik}^{(k)} f_{kj}^{(k)}$  and  $g^{(k+1)} := f_{kk}^{(k)} g^{(k)}$  are stored in the Turing machine in binary form. There is a problem with this strategy. Consider Example B.15 taken from Grötschel, Lovász, and Schrijver [212].

### Example B.15

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 \\ 1 & 1 & 1 & 2 & 0 \\ 1 & 1 & 1 & 1 & 2 \end{bmatrix}$$

Applying the update formula for this strategy gives  $f_{ii}^{(k)} = 2^{2^{k-1}}$  for  $i \geq k$  and  $g^{(k)} = 2^{(2^{(k-1)}-1)}$ . This follows from observing  $f_{kj}^{(k)} = 0$  for  $j > k$  which implies  $f_{ii}^{(k+1)} = f_{ii}^{(k)} f_{kk}^{(k)}$  for  $i \geq k+1$ . Then at the start of pivot  $k$  the diagonal elements  $a_{ii}^k$  are equal for all  $i \geq k$ , and the effect of pivot  $k$  is to square the diagonal elements for  $i \geq k+1$ . With this strategy the binary encoding of the data requires an exponential amount of space as a function of the size of the input matrix!

**Strategy 2:** Avoid fractions. Another strategy is to start with an integer matrix and at each step of Gaussian elimination pre-multiply rows  $i > k$  of matrix  $A^k$  by an integer  $c$  so that every element in column  $k$  is an integer multiple of the pivot element  $a_{kk}^{(k)}$ . The smallest element that can be selected, in general, is the pivot element itself, i.e.  $c = a_{kk}^{(k)}$ . Now consider an integer  $n \times n$  matrix where the diagonal elements are the integer  $x$ , the elements above the diagonal are zero, the elements below the diagonal are 1 except for row  $n$  where the elements are  $x$ . Then following this strategy, in matrix  $A^{(k)}$  element  $a_{nn}^{(k)}$  is order  $x^{x^{(k-1)}}$  and the binary encoding requires exponential space as a function of the size of the input matrix. An example for  $n = 5$  is given below.

### Example B.16

$$A = \begin{bmatrix} x & 0 & 0 & 0 & 0 \\ 1 & x & 0 & 0 & 0 \\ 1 & 1 & x & 0 & 0 \\ 1 & 1 & 1 & x & 0 \\ x & x & x & x & x \end{bmatrix}$$

**Strategy 3:** Store the rational number  $p/q$  as integers  $p$  and  $q$  and do addition as described in Strategy 1, but always bring  $p/q$  into *coprime form*. That is, make sure that the integers  $p$  and  $q$  are relatively prime. In this case the size of the numbers is a polynomial function of the input.

Let  $q^{(k+1)}$  for  $k = 1, \dots, r-1$  where  $r$  is the rank of  $A$  denote the value of the determinant of the  $k \times k$  submatrix of  $A^{(k+1)}$  defined by rows  $\{1, \dots, k\}$  and columns  $\{1, \dots, k\}$ . Define for  $i, j > k$ ,  $p_{ij}^{(k+1)}$  to be the determinant of the submatrix of  $A^{(k+1)}$  defined by rows  $\{1, \dots, k, i\}$  and columns  $\{1, \dots, k, j\}$ . Since the first  $k$  rows and columns of  $A^{(k+1)}$  form a diagonal matrix with determinant  $q^{(k+1)}$  it follows that

$$a_{ij}^{(k+1)} = \frac{p_{ij}^{(k+1)}}{q^{(k+1)}}, \quad i, j > k. \quad (\text{B.4})$$

When  $k = 0$ ,  $q^{(1)} := 1$  and  $p_{ij}^{(1)} := a_{ij}$ . Since row operations do not change the value of a determinant

$$a_{ij}^{(k+1)} = \det(A_{1,\dots,k,i}^{1,\dots,k,j})/\det(A_{1,\dots,k}^{1,\dots,k}), \quad i, j > k, \quad (\text{B.5})$$

where  $A_{1,\dots,k,i}^{1,\dots,k,j}$  is the submatrix of  $A$  induced by columns  $1, \dots, k, j$  and rows  $1, \dots, k, i$ . Therefore, if each  $a_{ij}^{(k+1)}$  is stored in coprime form it requires by Lemma B.4 at most size  $2(2(A) - n^2)$ . Each element in  $A^{(r)}$  for  $j \geq i$ , was at some iteration an  $a_{ij}^{(k+1)}$  for  $i, j > k$ ,  $k = 0, \dots, r-1$ . Therefore the size of  $A^{(r)}$  is a polynomial function of the size of  $A$ . Since exchanging columns and or rows can only change the sign of a determinant multiplication by permutation matrices  $P$  and  $Q$  has no effect on the size of the  $A^{(k)}$ .

It is easy to see that if  $A$  is  $m \times n$ , then Gaussian elimination requires at most  $O(\min\{m, n\}mn)$  arithmetic operations. The integers defining the rational number  $p/q$  can be put into coprime form using the Euclidean algorithm. The complexity of the Euclidean algorithm is  $O(\log_2(\max\{|p|, |q|\}))$ . Therefore with Strategy 3, Gaussian elimination is a polynomial algorithm with complexity  $O(\min\{m, n\}mn(A))$ . Since the size of the input affects the complexity Strategy 3 does not give a strongly polynomial algorithm. A relatively simple modification of Strategy 3 gives a strongly polynomial algorithm. This is due to Edmonds [136].

**Strategy 4:** The basic idea is to use (B.4) in the update formulas (13.1)-(13.2). That is, for  $i, j > k$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)} / a_{kk}^{(k)}) a_{kj}^{(k)}, \quad (\text{B.6})$$

$$= \frac{p_{ij}^{(k)}}{q^{(k)}} - \frac{p_{ik}^{(k)} / q^{(k)}}{p_{kk}^{(k)} / q^{(k)}} (p_{kj}^{(k)} / q^{(k)}) \quad (\text{B.7})$$

$$= \left( \frac{p_{kk}^{(k)} p_{ij}^{(k)} - p_{ik}^{(k)} p_{kj}^{(k)}}{q^{(k)}} \right) / p_{kk}^{(k)} \quad (\text{B.8})$$

By definition,  $q^{(k+1)}$  is the determinant of the  $k \times k$  submatrix of  $A^{(k+1)}$  defined by rows  $\{1, \dots, k\}$  and columns  $\{1, \dots, k\}$ . By definition,

$$q^{(k+1)} = p_{kk}^{(k)}. \quad (\text{B.9})$$

This and (B.8) give

$$p_{ij}^{(k+1)} = \frac{p_{kk}^{(k)} p_{ij}^{(k)} - p_{ik}^{(k)} p_{kj}^{(k)}}{q^{(k)}} \quad (\text{B.10})$$

Then (B.9)-(B.10) are used to update  $a_{ij}^{(k+1)}$  in (B.4). The *key* is that  $p_{ij}^{k+1}$  in (B.10) is *not* stored as a rational number since the denominator must divide the numerator because  $p_{ij}^{(k+1)}$  is the determinant of an integer matrix. This division counts as one arithmetic operation. Therefore  $a_{ij}^{(k+1)}$  is stored as two integers with size a polynomial function of the problem input.

From (13.6) and the argument given above, the size of  $L$  (and thus  $L^{-1}$ ) is a polynomial function of the size of  $A$ . Since  $U = L^{-1}PAQ$ , the upper triangular matrix  $U$  has size which is a polynomial function of the size of  $A$ . The fact that there is a solution to  $Ax = b$  with size a polynomial function of the size of  $A$  and  $b$  now follows immediately from (13.11)-(13.12). The only division required is integer division. Therefore, Gaussian elimination is strongly polynomial with complexity  $O(\min\{m, n\}mn)$ .

## B.8 EXERCISES

- B.1 Given the system  $Ax = b$  with  $A$  a rational  $m \times n$  matrix and  $b$  a rational  $m$  vector, show to construct an equivalent (in the sense that the new system has an identical solution set as the original system)  $A'x = b'$  where  $A'$  and  $b'$  are *integer* and the size of  $A', b'$  are polynomial functions of the size of  $A, b$ , respectively.
- B.2 Prove that upon completion of Algorithm B.2  $x[n + 1] = 2^n$ .
- B.3 Is it possible to encode the matrix in Example B.7 if the alphabet is restricted to  $\Omega = \{0, 1\}$ . If so, what additional, if any, assumptions are required.
- B.4 Consider the undirected graph  $G = (V, E)$  with vertex set  $V = \{1, \dots, n\}$  and edge set  $E$ . The decision problem is “decide if there is a path from vertex 1 to vertex  $n$ .” Express this problem in the formalism developed in Section B.3.
- B.5 Given undirected graph  $G = (V, E)$ , show that the problem of deciding the existence of a Hamiltonian cycle is in  $\mathcal{NP}$ . A Hamiltonian cycle is a cycle that includes every vertex exactly once.
- B.6 Prove that if  $\mathcal{L} \in \mathcal{P}$  then  $\mathcal{L} \in co\mathcal{NP}$ .
- B.7 Prove Lemma B.12. Explicitly define (specify the  $\delta$  function) the Turing machine you use in the proof.

- B.8 Explain why every formula in *CNF* form cannot be converted to a formula of Horn clauses by making the simple substitution  $x_i = \neg x_j$ .
- B.9 There is a subtle point left out of the proof of Theorem B.14. It turns out that determining satisfiability of a string of Horn clauses is easily reduced to testing the feasibility of a linear program which is a decision problem in  $\mathcal{P}$ . But Table B.1 is nothing more than a system of Horn clauses. Therefore, any problem in  $\mathcal{NP}$  reduces to feasibility of a linear program and  $\mathcal{P} = \mathcal{NP}$ ? Where is the flaw in the logic?
- B.10 Let  $\mathcal{L}_{TAUT}$  be the set of strings of clauses (not necessarily in CNF form) which are tautologies, i.e. true for every valuation of the variables. For example one such string of clauses is  $x_1 \vee \neg x_1$ . What is the implication of being able to show  $\mathcal{L}_{TAUT} \in \mathcal{P}$ ?

# C

---

## BASIC GRAPH THEORY

An *undirected graph* is a pair  $G = (V, E)$  where  $V$  is finite *vertex* set and  $E$  is a set of unordered pairs of elements from  $V$ . The elements of  $E$  are called *edges*. Graphs are represented by small dots or circles for the vertices in the plane and an edge is represented by a line connecting the vertices.

**Example C.1** Let  $G = (V, E)$  with

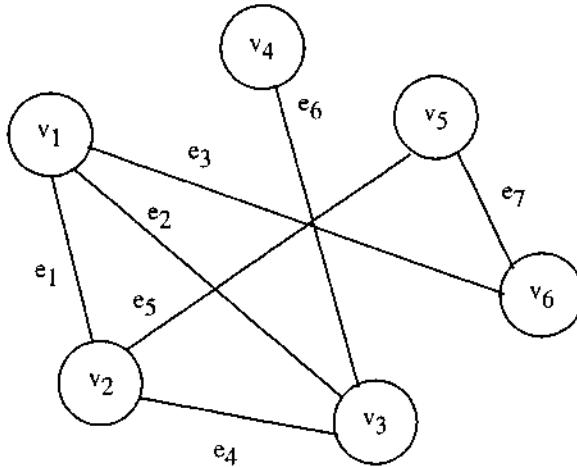
$$\begin{aligned}V &= \{v_1, v_2, v_3, v_4, v_5, v_6\} \\E &= \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_6\}, \{v_2, v_3\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_5, v_6\}\}.\end{aligned}$$

A representation of this graph in the plane is given in Figure C.1.

If  $v_1$  and  $v_2$  are two vertices of a graph and if  $\{v_1, v_2\}$  is in the edge set, then  $v_1$  and  $v_2$  are *adjacent vertices* and  $v_1, v_2$  are *incident* with the edge  $\{v_1, v_2\}$ . An edge with both vertices identical is called a *loop* and an edge with distinct vertices is called a *link*. A *simple graph* is one where no edges are loops and no two links are incident to the same set of vertices.

A *walk* from vertex  $v_0$  to vertex  $v_k$  in  $G$ , is a finite non-null sequence  $W = v_0e_1v_1e_2v_2 \cdots e_kv_k$  whose terms are alternating vertices and edges, and for  $1 \leq i \leq k$ , vertices  $v_{i-1}$  and  $v_i$  are incident to edge  $e_i$ . Vertex  $v_0$  is the *starting* or *origin* vertex and vertex  $v_k$  is the *ending* or *terminus* vertex. If the origin and terminus vertex are identical it is a *closed walk*. If the edges in a walk  $W$ , are distinct it is called a *trail*. If the vertices in a trail are distinct it is called a *path*. A graph  $G$  is a *connected graph* if, for every pair of vertices  $v_i, v_j$ , there is a path from vertex  $v_i$  to vertex  $v_j$ . In the graph of Example C.1, an example path is

$$v_6e_7v_5e_5v_2e_1v_1e_2v_3.$$

**Figure C.1** Graph of Example C.1

A *cycle* is a closed walk with distinct edges and vertices (except for the origin and terminus). A graph is an *acyclic graph* if it contains no cycles. A *tree* is a connected acyclic graph. An acyclic graph is also called a *forrest*.

The graph  $H = (\hat{V}, \hat{E})$  is a *subgraph* of  $G = (V, E)$  if  $\hat{V} \subseteq V$  and  $\hat{E} \subseteq E$ . A subgraph  $H = (\hat{V}, \hat{E})$  of  $G = (V, E)$  is a *spanning tree* if  $\hat{V} = V$  and  $H$  is a tree. If there is a weight  $c_e$ , assigned to each edge  $e \in E$  the *minimum spanning tree* is the spanning tree of minimum weight. A subgraph  $H = (\hat{V}, \hat{E})$  of  $G = (V, E)$  is a *clique* if every vertex in  $\hat{V}$  is adjacent to every other vertex in  $\hat{V}$ .

For  $S \subseteq V$ , let  $\gamma(S) \subset E$  denote all edges of  $E$  with both endpoints in  $S$ ,  $\delta(S) \subseteq E$  the set of edges with exactly one endpoint in  $S$  and  $\Gamma(S)$  the set of vertices adjacent to at least one vertex in  $S$ . For  $E' \subset E$  define  $x(E') = \sum_{e \in E'} x_e$ .

A *directed graph* is a pair  $G = (V, \mathcal{A})$  where  $V$  is a vertex set and  $\mathcal{A}$  is a set of ordered pairs of elements from  $V$ . The elements of  $\mathcal{A}$  are called *arcs*. If  $(v_1, v_2) \in \mathcal{A}$  vertex  $v_1$  is called the *tail* and vertex  $v_2$  is called the *head*. A *directed path* from vertex  $v_0$  to vertex  $v_k$  in  $G$ , is a finite non-null sequence  $v_0 a_1 v_1 a_2 v_2 \cdots a_k v_k$  whose terms are alternating vertices and arcs, and for  $1 \leq i \leq k$ ,  $a_i = (v_{i-1}, v_i)$ . If the edges in a directed walk are distinct it is called a *directed trail*. If the vertices in a directed trail are distinct it is called a *directed path*. A *directed cycle* is closed directed walk with at least one arc and no repeated arcs or vertices (except the start and end vertex).

If  $G = (V, \mathcal{A})$  is a directed graph and  $S \subseteq V$ , define

$$\delta(S) := \{(v_1, v_2) \in \mathcal{A} \mid v_1 \in S, v_2 \in V \setminus S, \text{ or } v_2 \in S, v_1 \in V \setminus S\}.$$

The set of arcs in  $\delta(S)$  is a *cut* or *cutset* of  $G$ . If  $s \in S$  and  $t \in V \setminus S$  the set  $\delta(S)$  is called an *s-t cut* or *s-t cutset*. Clearly, deleting the arcs in an  $s - t$  cut eliminates all directed paths from  $s$  to  $t$ .

A *Leontief directed hypergraph* is a pair  $G = (V, \mathcal{H})$  where  $V$  is a finite set of vertices and  $\mathcal{H}$  a finite set of *hyperarcs*. Each hyperarc is an ordered pair  $(J, k)$ , where  $J \subseteq V$ ,  $k \in V \setminus J$ . Vertex  $k$  is the *head* of the hyperarc and  $J$  is the *tail set*. We allow *tailless* or *source* hyperarcs  $(\emptyset, k)$  directed into  $k$ , and *headless* or *sink* hyperarcs  $(J, \emptyset)$  directed out of tail set  $J$ . The coefficient associated with the head of a hyperarc is +1. *Tail weights*  $\{a_j[J, k] \mid j \in J\}$  give the magnitudes of corresponding negative coefficients at tails  $j \in J$ . Models can have several “parallel” hyperarcs with different tail weights connecting the same  $(J, k)$  pair. In the interest of notational simplicity, we assume each  $(J, k)$  pair identifies a unique hyperarc. Parallel hyperarcs are easily made unique through the introduction of artificial vertices.

A *directed walk* in Leontief directed hypergraph  $G$  from vertex  $v_1$  to vertex  $v_{k+1}$  is defined by the non-null sequence  $v_1 e_1 v_2 e_2 v_3, \dots, e_k v_{k+1}$  whose terms are alternately vertices and hyperarcs such that  $e_i = (J_i, v_{i+1})$  and  $v_i \in J_i$ . Similarly for directed trail and path. A directed walk with distinct hyperarcs from vertex  $v_1$  to vertex  $v_{k+1}$  is a *directed cycle* if  $v_1 = v_{k+1}$  and if no other vertices are repeated. A Leontief directed hypergraph with no directed cycles is *acyclic*.

For more on graph theory see *Graphs and Hypergraphs* by Berge [50], *Graph Theory An Introductory Course* by Bollobás [64], and *Graph Theory with Applications* by Bondy and Murty [65].

# D

---

## SOFTWARE AND TEST PROBLEMS

The importance of a large readily available set of test problems cannot be overstated. The emergence of test banks of problems of large scale realistic problems has greatly advanced mathematical programming research. This has allowed researchers to make consistent and reproducible comparisons of algorithms. Many test problems are available through the Internet, either through FTP or WWW servers. Here is a partial list.

**NETLIB:** This is a standard set of linear programming test problems. These are often used to test and compare simplex and interior point algorithms.

<http://www.mcs.anl.gov/home/otc/Guide/TestProblems/LPtest/index.html>

This Web site includes the standard set, the infeasible library and the Kensington problems.

**The Rice MIP Library:** An excellent set of mixed integer linear programming problems including the famous Crowder, Johnson, and Padberg test problems.

<http://www.caam.rice.edu/~bixby/miplib/miplib3.html>

**Traveling Salesman Problems:**

<http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html>

**Vehicle Routing Problems:**

[http://dmawww.epfl.ch/~rochat/rochat\\_data/solomon.html](http://dmawww.epfl.ch/~rochat/rochat_data/solomon.html)

**Other Problems:** For dynamic lot sizing and others.

<ftp://gsbkip.uchicago.edu/Files/faculty/martin/iptest/>

Many of the test problems at the Web sites listed come from actual applications. However, in order to do a serious statistical analysis of an algorithm it may be necessary to generate random variations of a problem. See Greenberg [207] for a description of RANDMOD which is designed to perform controlled randomization on a linear program. Lin and Rardin [289] discuss setting up a controlled experimental design for comparing algorithms. If a random number generator is used in experimental testing it is important to use one that is portable and as compiler independent as possible. See Bratley, Fox, and Schrage [73] and Degraeve and Schrage [119].

Comparing the efficiency of algorithms on test problems is difficult. CPU time is often used. However, there are problems with using CPU time as a benchmark. See, for example, Dowd [130]. One can also use operation counts, see Ahuja and Orlin [8].

Finally, it is crucial to perform and report computational experiments so that they can be replicated. See Jackson et al. [241] for guidelines proposed by COAL – the Committee on Algorithms. These issues are also often discussed in COAL newsletters published by the Mathematical Programming Society.

linear programs. In Chapter 8,  $\Gamma$  represents a trajectory of solutions to barrier problems parameterized in  $\mu$ , the barrier parameter.

An objective function value is represented by  $z_0$  and  $z_0(b)$  is an optimal value function. An upper case  $M$  or “big M” is used for a large constant in fixed charge constraints.

The symbol  $|$  is “such that” and used to define sets. For example,  $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ . The notation  $a_1 \leftarrow a_2$  means replace the memory location of  $a_1$  with the quantity in the location of  $a_2$ . Throughout the text, in formal statements of algorithms, we replace  $=$  with  $\leftarrow$  if the left hand side is replaced or updated by the right hand side.

Table E.1 Notation

Symbol	Meaning	Defined
s.t.	subject to	6
M	big M	15
$\text{proj}_y(P)$	projection of a polyhedron into the $y$ variable space	36
$\mathbb{R}$	set of real numbers	36
$\doteq$	definitional equality	36
$\Rightarrow$	implication	63
$z_0(b)$	optimal value function of right hand side vector $b$	74
$e^i$	$i$ th unit vector	89
$\mathbb{Z}$	set of integers	105
$\mathbb{Z}_+$	set of nonnegative integers	105
$a \equiv b \pmod{\theta}$	$\theta$ divides $a - b$	106
$[a]$	largest integer $\leq a$	106
$\gcd(a_1, a_2)$	greatest common divisor of $a_1$ and $a_2$	106
$\leftarrow$	assign a value to a memory location	107
$[a]$	smallest integer $\geq a$	112
$A_B$	columns of $A$ indexed by $B$	144
$\text{argmin}$	the element of a function domain which minimizes the function	145
$x_{B_i^\top}$	variable indexed by $i$ th component of basis index set $B$	148
$A^{-\top}$	inverse of $A^\top$	192
$e$	a vector of 1s of appropriate dimension	228
$\det(A_B)$	determinant of matrix $A_B$	239
$\text{adj}(A_B)$	adjoint matrix of matrix $A_B$	239
$\mu$	barrier function parameter	262
$z_0(*)$	the optimal solution value of problem (*)	394
$\text{aff}(x^1, \dots, x^n)$	affine hull of the set $\{x^1, \dots, x^n\}$	636
$\text{cone}(x^1, \dots, x^n)$	conic hull of the set $\{x^1, \dots, x^n\}$	636
$\text{conv}(x^1, \dots, x^n)$	convex hull of the set $\{x^1, \dots, x^n\}$	636
$\ x\ $	Euclidean norm of $x$	651
$O(*)$	algorithm complexity	657
$\Omega$	a finite set of symbols	662
$\mathcal{P}$	polynomial class of languages	663
$\mathcal{NP}$	nondeterministic polynomial class of languages	664
$G = (V, E)$	undirected graph	677
$G = (V, \mathcal{A})$	directed graph	678
$\gamma(S)$	edges of $E$ with both endpoints in $S$	678
$x(E')$	$\sum_{e \in E'} x_e$	678
$\delta(S)$	cutset	679
	such that	684

---

## REFERENCES

- [1] I. Adler and S. Cosares. A strongly polynomial algorithm for a special class of linear programs. *Operations Research*, 39:955–960, 1991.
- [2] I. Adler, N. K. Karmarkar, M. G. C. Resende, and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar’s algorithm. *ORSA Journal of Computing*, 1:84–106, 1989.
- [3] I. Adler, N. K. Karmarkar, M. G. C. Resende, and G. Veiga. An implementation of Karmarkar’s algorithm for linear programming. *Mathematical Programming*, 44:297–325, 1989.
- [4] I. Adler and R.D. C. Monterio. A geometric view of parametric linear programming. *Algorithmica*, 8:161–176, 1992.
- [5] P. B. Afentakis, B. Gavish, and U. Karmarkar. Computationally efficient optimal solutions to the lot-sizing problem in multistage assembly systems. *Management Science*, 30:222–239, 1984.
- [6] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974.
- [7] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [8] R.K. Ahuja and J.B. Orlin. Use of representative operation counts in computational testing of algorithms. *INFORMS Journal on Computing*, 8:318–330, 1996.
- [9] R. Anbil, E. Gelman, B. Patty, and R. Tanga. Recent advances in crew-pairing optimization at American Airlines. *Interfaces*, 21:62–74, 1991.
- [10] R. Anbil, R. Tanga, and E.L. Johnson. A global approach to crew-pairing optimization. *IBM Systems Journal*, 31:71–78, 1992.
- [11] E.D. Andersen and K.D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71:221–245, 1995.

- [12] D. R. Anderson, D. J. Sweeney, and T. A. Williams. *An Introduction to Management Science*. West Publishing, St. Paul, MN, sixth edition, 1991.
- [13] K. Anstreicher. On long step path following and SUMT for linear and quadratic programming. *SIAM Journal on Optimization*, 6:33–46, 1996.
- [14] K. Anstreicher and R.A. Bosch. Long steps in an  $O(n^3l)$  algorithm for linear programming. *Mathematical Programming*, 54:251–265, 1992.
- [15] D. Applegate, R. Bixby, V., and W. Cook. Finding cuts in the TSP. DIMACS Center 95.05, Rutgers University, 1995.
- [16] E.P. Armendáriz and S. J. McAdam. *Elementary Number Theory*. MacMillan, New York, 1980.
- [17] R.D. Armstrong and P. Sinha. Improved penalty calculations for a mixed integer branch-and-bound algorithm. *Mathematical Programming*, 6:212–223, 1974.
- [18] A. Bachem and W. Kern. *Linear Programming Duality An Introduction to Oriented Matroids*. Springer-Verlag, Berlin, Germany, 1992.
- [19] H.C. Bahl and S. Zionts. Multi-item scheduling by Benders decomposition. School of Management Working Paper 522, University of New York at Buffalo, 1983.
- [20] E. Balas. Disjunctive programming: Properties of the convex hull of feasible points. Management Science Research Group MSRR-348, Carnegie Mellon University, 1974.
- [21] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8:146–164, 1975.
- [22] E. Balas. Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.
- [23] E. Balas. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic and Discrete Methods*, 6:466–486, 1985.
- [24] E. Balas. Projection with a minimal system of inequalities. Management Science Research Group MSRR-585, Carnegie Mellon University, 1992.
- [25] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

- [26] E. Balas, S. Ceria, and G. Cornuéjols. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.
- [27] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42:1229–1246, 1996.
- [28] E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal of Applied Mathematics*, 23:61–69, 1972.
- [29] E. Balas and W. Pulleyblank. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13:495–516, 1983.
- [30] E. Balas and E. Zemel. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics*, 34:119–148, 1978.
- [31] M. Balinski. Fixed cost transportation problems. *Naval Research Logistics Quarterly*, 8:41–54, 1961.
- [32] M. Balinski. Integer programming: Methods, uses, computation. *Management Science*, 12:253–313, 1965.
- [33] J. L. Balintfy. Menu planning by computer. *Communications of the ACM*, 7:255–259, 1964.
- [34] I. Barany, T. J. Van Roy, and L. A. Wolsey. Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30:1255–1261, 1984.
- [35] I. Barany, T.J. Van Roy, and L. A. Wolsey. Strong formulations for multi-item capacitated lot sizing. *Mathematical Programming Study*, 22:32–43, 1984.
- [36] E. R. Barnes. A variation of Karmarkar's linear programming algorithm for solving linear programming problems. *Mathematical Programming*, 16:174–182, 1986.
- [37] C. Barnhart, C.A. Hane, and P. H. Vance. Integer multicommodity flow problems. Technical report, Auburn University, 1995.
- [38] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. In J.R. Birge and K.G. Murty, editors, *Mathematical Programming State of the Art 1994*, pages 186–207. University of Michigan and Braun-Brumfield, Inc., Ann Arbor, Michigan, 1994.

- [39] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.
- [40] R.H. Bartels and G.H. Golub. The simplex method of linear programming using LU decomposition. *Communications of the ACM*, 12:266–268, 275–278, 1969.
- [41] D.O. Bausch, G.G. Brown, D.R. Hundley, S.H. Rapp, and R.E. Rosenthal. Mobilizing Marine Corps officers. *Interfaces*, 21:26–38, 1991.
- [42] D. Bayer and J. C. Lagarias. The nonlinear geometry of linear programming I. affine and projective scaling trajectories. *Transactions of the American Mathematical Society*, 314:499–526, 1989.
- [43] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, New York, second edition, 1990.
- [44] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming Theory and Algorithms*. John Wiley & Sons, Inc., New York, 1993.
- [45] E. M. L. Beale. Cycling in the dual simplex algorithm. *Naval Research Logistics Quarterly*, 2:269–276, 1955.
- [46] E. M. L. Beale and R. E. Small. Mixed integer programming by a branch and bound technique. In W. A. Kalenich, editor, *Proc. IFIP Congress, Vol. 2*, pages 450–451, Washington D. C. and London, 1965. Spartan Press and Macmillan.
- [47] E. M. L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. In J. Lawrence, editor, *Proceedings of the Fifth International Conference on Operational Research*, pages 447–454, London, 1969. Tavistock Publications.
- [48] J. C. Bean, C. E. Noon, and G. J. Salton. Asset divestiture at Homart Development. *Interfaces*, 17:48–64, 1987.
- [49] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
- [50] C. Berge. *Graphs and Hypergraphs*. North-Holland, Amsterdam, 1973.
- [51] O. Bilde and J. Krarup. Sharp lower bounds and efficient algorithms for the simple plant location problem. *Annals of Discrete Mathematics*, 1:79–97, 1977.

- [52] S.C. Billups and M.C. Ferris. Convergence of an infeasible-interior-point algorithm from arbitrary positive starting points. *SIAM Journal on Optimization*, 6:316–325, 1996.
- [53] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, New York, 1997.
- [54] R. E. Bixby. Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, 4:267–284, 1992.
- [55] R. E. Bixby. Progress in linear programming. *ORSA Journal on Computing*, 6:15–22, 1994.
- [56] R.E. Bixby. Kuratowski's and Wagner's theorems for matroids. *Journal of Combinatorial Theory (B)*, 22:31–53, 1977.
- [57] R.E. Bixby and W.H. Cunningham. Converting linear programs to network problems. *Mathematics of Operations Research*, 14:321–357, 1976.
- [58] R.E. Bixby and M.J. Saltzman. Recovering an optimal LP basis from an interior point solution. Department of Mathematical Sciences Technical Report #607, Rich University, 1992.
- [59] R.E. Bixby and D.K. Wagner. A note on detecting simple redundancies in linear systems. *Operations Research Letters*, 6:15–17, 1987.
- [60] C.E. Blair. A closed-form representation of mixed-integer program value functions. *Mathematical Programming*, 71:127–136, 1995.
- [61] C.E. Blair and R.G. Jeroslow. The value function of an integer program. *Mathematical Programming*, 23:237–273, 1982.
- [62] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2:103–107, 1977.
- [63] R.G. Bland, D. Goldfarb, and M.J. Todd. The ellipsoid method: A survey. *Operations Research*, 29:1039–1091, 1981.
- [64] B. Bollobás. *Graph Theory An Introductory Course*. Springer-Verlag, New York, 1979.
- [65] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North-Holland, New York, Amsterdam, Oxford, 1976.
- [66] E. A. Boyd. Generating Fenchel cutting planes for knapsack polyhedra. *SIAM Journal on Optimization*, 3:734–750, 1993.

- [67] E. A. Boyd. Fenchel cutting planes for integer programs. *Operations Research*, 42:53–64, 1994.
- [68] E. A. Boyd. On the convergence of Fenchel cutting planes in mixed-integer programming. *SIAM Journal on Optimization*, 5:421–435, 1995.
- [69] E. A. Boyd. Resolving degeneracy in combinatorial linear programs: Steepest edge, steepest ascent, and parametric ascent. *Mathematical Programming*, 68:155–168, 1995.
- [70] G.H. Bradley. Algorithms for Hermite and Smith normal matrices and linear diophantine equations. *Mathematics of Computation*, 25:897–907, 1971.
- [71] G.H. Bradley, G.G. Brown, and G.W. Graves. Design and implementation of large scale primal transshipment algorithms. *Management Science*, 24:1–34, 1977.
- [72] S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, Reading, MA, 1977.
- [73] P.B. Bratley, B. Fox, and L. Schrage. *A Guide to Simulation*. Springer-Verlag, New York, 1987.
- [74] A.L. Brearley, G. Mitra, and H.P. Williams. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Mathematical Programming*, 5:54–83, 1975.
- [75] A. Brøndsted. *Convex Polytopes*. Springer Verlag, New York, 1983.
- [76] S. Cabay and T.P.L. Lam. Congruence techniques for the exact solution of integer systems of linear equations. *ACM Transactions on Mathematical Software*, 3:386–397, 1977.
- [77] P. Cacioppi. New anticycling strategies for the simplex method. Computer science masters thesis, University of Chicago, 1996.
- [78] P. Camion. Charactérisation des matrices unimodulaires. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 5:181–190, 1963.
- [79] P. Camion. Characterizations of totally unimodular matrices. *Proceedings of the American Mathematical Society*, 16:1068–1073, 1965.
- [80] J. D. Camm, A. S. Rauri, and S. Tsubakitani. Cutting big M down to size. *Interfaces*, 20:61–66, 1990.

- [81] J.D. Camm, T.E. Chorman, F.A. Dill, J.R. Evans, D.J. Sweeney, and G.W. Wegryn. Blending OR/MS, judgement, and GIS: Restructuring P&G's supply chain. *Interfaces*, 27:128–142, 1997.
- [82] J.D. Camm and J.R. Evans. *Management Science Modeling, Analysis and Interpretation*. South-Western College Publishing, Cincinnati, OH, 1996.
- [83] J.F. Campbell and A. Langevin. The snow disposal assignment problem. *Journal of the Operational Research Society*, 48:919–929, 1995.
- [84] T. J. Carpenter, I. J. Lustig, J. M. Mulvey, and D. F. Shanno. Higher order predictor-corrector interior point methods with application to quadratic objectives. *SIAM Journal on Optimization*, 3:696–725, 1993.
- [85] S. Ceria, C. Codier, H. Marchand, and L.A. Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81:201–214, 1998.
- [86] L.G. Chalmet, R.L. Francis, and P.B. Saunders. Network models for building evacuation. *Management Science*, 28:86–105, 1982.
- [87] S.F. Chang and S.T. McCormick. A hierarchical algorithm for making sparse matrices sparser. *Mathematical Programming*, 56:1–30, 1992.
- [88] A. Charnes. Optimality and degeneracy in linear programming. *Econometrica*, 20:160–170, 1952.
- [89] J.W. Chinneck and E.W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3:157–168, 1991.
- [90] I. C. Choi, C. L. Monma, and D. F. Shanno. Further development of a primal-dual interior point method. *ORSA Journal on Computing*, 2:304–311, 1989.
- [91] T.J. Chou and G.E. Collins. Algorithms for the solution of systems of linear diophantine equations. *SIAM Journal Computing*, 11:687–708, 1982.
- [92] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:137–179, 1973.
- [93] V. Chvátal. *Linear Programming*. W.H. Freeman and Company, New York, 1988.
- [94] F.E. Clark. Remark on the constraint sets in linear programming. *American Mathematical Monthly*, 68:351–352, 1961.

- [95] A. Claus. A new formulation for the travelling salesman problem. *SIAM Journal on Algebraic and Discrete Methods*, 5:21–25, 1984.
- [96] S. D. Conte and C. de Boor. *Elementary Numerical Analysis an Algorithmic Approach*. McGraw-Hill, New York, 1972.
- [97] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings 3rd Annual ACM Symposium of Theory of Computing*, pages 151–158. Association For Computing Machinery, 1971.
- [98] W. Cook. Operations that preserve total dual integrality. *Operations Research Letters*, 2:31–35, 1983.
- [99] W. Cook, L. Lovász, and A. Schrijver. A polynomial-time test for total dual integrality in fixed dimension. *Mathematical Programming Study*, 22:64–69, 1984.
- [100] W. Cook, T. Rutherford, H. E. Scarf, and D. Shallcross. An implementation of the generalized basis reduction algorithm for integer programming. *ORSA Journal on Computing*, 5:206–212, 1993.
- [101] G. Cornuéjols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23:789–810, 1977.
- [102] H.P. Crowder and J.M. Hattingh. Partially normalized pivot selection in linear programming. Technical Report RC 4918, IBM, 1974.
- [103] H.P. Crowder, E. L. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.
- [104] W.H. Cunningham. A network simplex method. *Mathematical Programming*, 11:105–116, 1976.
- [105] W.H. Cunningham. Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 4:196–208, 1979.
- [106] W.H. Cunningham. Minimum cuts, modular functions, and matroid polyhedra. *Networks*, 15:205–215, 1985.
- [107] A.R. Curtis and J.K. Reid. On the automatic scaling of matrices for gaussian elimination. *Journal of the Institute of Mathematics and its Applications*, 10:118–124, 1972.
- [108] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

- [109] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin eliminations and its dual. *Journal of Combinatorial Theory*, 14:288–297, 1973.
- [110] G. B. Dantzig and W. Orchard-Hays. The product form of the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, pages 64–67, 1954.
- [111] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5:183–195, 1955.
- [112] G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29:767–778, 1961.
- [113] G.B. Dantzig. Reminiscences about the origins of linear programming. *Operations Research Letters*, 1:43–48, 1982.
- [114] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. Solution of a large-scale travelling salesman problem. *Operations Research*, 2:393–410, 1954.
- [115] G.B. Dantzig, D.R. Fulkerson, and S.M. Johnson. On a linear programming, combinatorial approach to the traveling salesman problem. *Operations Research*, 7:58–66, 1959.
- [116] G.B. Dantzig and R.M Van Slyke. Generalized upper bounding techniques. *Journal Computer System Science*, 1:213–226, 1967.
- [117] G.B. Dantzig and M.N. Thapa. *Linear Programming 1: Introduction*. Springer, New York, 1997.
- [118] A.C. Day. *FORTRAN Techniques with Special Reference to Non-Numerical Applications*. Cambridge University Press, Cambridge, 1972.
- [119] Z. Degraeve and L. Schrage. Should I use a portable generator in an emergency. Technical report, Katholieke Universiteit Lueven, 1997.
- [120] J. Desrochers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.E. Ball, T.L. Magnanti, and C. Monma and G.L. Nemhauser, editors, *Handbook in Operations Research and Management Science, Volume 8: Network Routing*, pages 35–140. Elsevier, Amsterdam, 1995.
- [121] M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23:1–13, 1989.

- [122] B.L. Dietrich and L.F. Escudero. Coefficient reduction for knapsack-like constraints in 0/1 programs with variable upper bounds. *Operations Research Letters*, 9:9–14, 1990.
- [123] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Mathematics Doklady*, 8:674–675, 1967.
- [124] I. I. Dikin. On the convergence of an iterative process. *Upravlyayemye Sistemi*, 12:54–60, 1974.
- [125] L. L. Dines. Systems of linear inequalities. *Annals of Mathematics, Second Series*, 20:191–199, 1918.
- [126] M. E. Doherty. Special structures in integer programming, Volume II. Technical report, Procter & Gamble, 1990.
- [127] P. D. Domich, P.T. Boggs, J. E. Rogers, and C. Witzgall. Optimizing over 3-dimensional subspaces in an interior point method for linear programming. *Linear Algebra and its Applications*, 152:315–342, 1989.
- [128] P. D. Domich, R. Kannan, and L. E. Trotter. Hermite normal form computation using modulo determinant arithmetic. *Mathematics of Operations Research*, 12:50–59, 1987.
- [129] J.J. Dongarra, F.G. Gustavson, and A. Karp. Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Review*, 26:91–112, 1984.
- [130] K. Dowd. *High Performance Computing*. O'Reilly & Associates, Inc., Sebastopol, CA, 1993.
- [131] K. Dowd. Optimizing with pre- and post-compilers. *Byte*, 19:82–83, 1994.
- [132] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, 1986.
- [133] R. J. Duffin. On Fourier's analysis of linear inequality systems. *Mathematical Programming Study*, 1:71–95, 1974.
- [134] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operations Research*, 54:7–22, 1991.
- [135] B. P. Dzielinski and R. E. Gomory. Optimal programming of lot sizes. *Management Science*, 11:874–890, 1965.

- [136] J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards (B)*, 71:241–245, 1967.
- [137] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.
- [138] M. A. Efronymson and T. L. Ray. A branch and bound algorithm for plant location. *Operations Research*, 14:361–368, 1966.
- [139] G. D. Eppen and R. K. Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35:832–848, 1992.
- [140] G. D. Eppen, R.K. Martin, and L. Schrage. A mixed-integer programming model for evaluating distribution configurations in the steel industry. *Computers and Operations Research*, 13:575–586, 1986.
- [141] G.D. Eppen, F.J. Gould, and C.P. Schmidt. *Introductory Management Science*. Prentice Hall, Englewood Cliffs, NJ, 1993.
- [142] R.E. Erickson, C.L. Monma, and A.F. Veinott, Jr. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12:634–664, 1987.
- [143] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26:992–1009, 1978.
- [144] J.R. Evans. A combinatorial equivalence between a class of multicommodity flow problems and the capacitated transporation problem. *Mathematical Programming*, 10:401–404, 1976.
- [145] H. Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11:399–417, 1963.
- [146] S. Fang and S. Puthenpura. *Linear Optimization and Extensions: Theory and Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [147] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. SIAM, Philadelphia, PA, 1968.
- [148] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.
- [149] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

- [150] M. L. Fisher, R. Jaikumar, and L.N. Van Wassenhove. A multiplier adjustment method for the generalized assignment problem. *Management Science*, 32:1095–1103, 1986.
- [151] M.M. Flood. The traveling salesman problem. *Operations Research*, 4:61–75, 1956.
- [152] L.R. Ford, Jr. and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, N.J., 1962.
- [153] J. J. H. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming*, 57:341–374, 1992.
- [154] J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin. Practical solution of large mixed integer programming problems with UMPIRE. *Management Science*, 20:736–773, 1974.
- [155] J. J. H. Forrest and J. Tomlin. Updating triangular factors of the basis to maintain sparsity in the product-form simplex method. *Mathematical Programming*, 2:263–278, 1972.
- [156] J. J. H. Forrest and J. Tomlin. Implementing interior point linear programming methods in the optimization subroutine library. *Mathematical Programming Systems RJ 7400* (69202), IBM, 1990.
- [157] J. J. H. Forrest and J. A. Tomlin. Implementing the simplex method for the optimization subroutine library. *IBM Systems Journal*, 31:11–25, 1992.
- [158] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL A Modeling Language for Mathematical Programming*. Scientific Press, San Francisco, CA, 1993.
- [159] J. B. J. Fourier. Solution d'une question particulière du calcul des inégalités. *Oeuvres II Paris*, pages 317–328, 1826.
- [160] R.M. Freund. Polynomial-time algorithms for linear programming based only on primal affine scaling and projected gradients of a potential function. *Mathematical Programming*, 51:203–222, 1991.
- [161] K. R. Frisch. The logarithmic potential method for convex programming. Unpublished manuscript, University of Oslo, 1955.
- [162] M.A. Frumkin. An application of modular arithmetic to the construction of algorithms for solving systems of linear equations. *Soviet Mathematics Doklady*, 17:1165–1168, 1976.

- [163] M.A. Frumkin. Polynomial time algorithms in the theory of linear diophantine equations. In M. Karpinski, editor, *Fundamentals of Computation Theory (Lecture Notes in Computer Science 56)*, pages 386–392. Springer, New York, 1977.
- [164] D.R. Fulkerson and P. Wolfe. An algorithm for scaling matrices. *SIAM Review*, 4:142–146, 1962.
- [165] C. B. Garcia and W. I. Zangwill. *Pathways to Solutions, Fixed Points, and Equilibria*. Prentice-Hall, Englewood Cliffs, N. J., 1981.
- [166] M. R. Garey and D. S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.
- [167] R.S. Garfinkel and G.L. Nemhauser. *Integer Programming*. John Wiley & Sons, New York, 1972.
- [168] S.I. Gass. *Linear Programming*. McGraw-Hill, New York, fourth edition, 1975.
- [169] J. von zur Gathen and M. Sieveking. Weitere zum erfüllungsproblem polynomial äquivalente kombinatorische aufgaben. In E. Specker and V. Strassen, editors, *Komplexität von Entscheidungsproblemen (Lecture Notes in Computer Science 43)*, pages 49–71. Springer, Heidelberg, 1976.
- [170] J. von zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalaities. *Proceedings of the American Mathematical Society*, 72:155–158, 1978.
- [171] J. M. Gauthier and G. Ribière. Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming*, 12:26–47, 1977.
- [172] A. M. Geoffrion. Elements of large-scale mathematical programming. *Management Science*, 16:652–691, 1970.
- [173] A. M. Geoffrion. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.
- [174] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.
- [175] A. M. Geoffrion and G.W. Graves. Multicommodity distribution system design by Benders decompositon. *Management Science*, 20:822–844, 1974.

- [176] A. M. Geoffrion and R. Marsten. Integer programming: A framework and state-of-the-art survey. *Management Science*, 18:465–491, 1972.
- [177] A. George, M.T. Heath, and J. Liu. Parallel Cholesky factorization on a shared-memory multiprocessor. *Linear Algebra and Its Applications*, 77:165–187, 1986.
- [178] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, N.J., 1981.
- [179] A. George and J. W.-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.
- [180] A. Ghouila-Houri. Caractérisation des matrices totalement unimodulaires. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences (Paris)*, 254:1192–1194, 1962.
- [181] F.R. Giles and W.R. Pulleyblank. Total dual integrality and integer polyhedra. *Linear Algebra and Its Applications*, 25:191–196, 1979.
- [182] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming*, 36:183–209, 1986.
- [183] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.
- [184] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem - Part II. *Operations Research*, 11:863–888, 1963.
- [185] J. Gleeson and J. Ryan. Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing*, 2:61–63, 1990.
- [186] F. Glover, J. Hultz, D. Klingman, and J. Stutz. Generalized networks: A fundamental computer-based planning tool. *Management Science*, 24:1209–1220, 1978.
- [187] J.-L. Goffin and J.-Ph. Vial. Cutting planes and column generation techniques with the projective algorithm. *Journal of Optimization Theory and Applications*, 65:409–429, 1990.
- [188] D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most  $nm$  pivots and  $O(n^2m)$  time. *Mathematical Programming*, 47:353–365, 1990.

- [189] D. Goldfarb and J.K. Reid. A practicable steepest-edge simplex algorithm. *Mathematical Programming*, 12:361–371, 1977.
- [190] A. J. Goldman and A. W. Tucker. Theory of linear programming. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 53–97. Princeton University Press, Princeton, NJ, 1956.
- [191] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin American Mathematical Society*, 64:275–278, 1958.
- [192] R. E. Gomory. Some polyhedra related to combinatorial problems. *Linear Algebra and Its Applications*, 2:451–558, 1969.
- [193] R.E. Gomory. An algorithm for integer solutions to linear programs. In R. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw Hill, New York, 1963.
- [194] J. Gondzio. Splitting dense columns of constraint matrix in interior point methods for large scale linear programming. *Optimization*, 24:285–297, 1992.
- [195] J. Gondzio. Presolve analysis of linear programs prior to applying an interior point method. Section of Management Studies 1994.3, University of Geneva, 1994.
- [196] C. Gonzaga. Interior point algorithms for linear programming problems with inequality constraints. Technical Report ES-140/88, COPPE-Federal University of Rio de Janeiro, 1988.
- [197] C. C. Gonzaga. An algorithm for solving linear programming problems in  $O(n^3l)$  operations. In N. Megiddo, editor, *Progress in Mathematical Programming Interior-Point and Related Methods*, pages 1–28. Springer-Verlag, New York and Berlin, 1989.
- [198] C.C. Gonzaga. Polynomial affine algorithms for linear programming. *Mathematical Programming*, 49:7–21, 1990.
- [199] C.C. Gonzaga. Large step path-following methods for linear programming, Part I: barrier function methods. *SIAM Journal on Optimization*, 1:268–279, 1991.
- [200] C.C. Gonzaga. Large step path-following methods for linear programming, Part II: potential reduction method. *SIAM Journal on Optimization*, 1:280–292, 1991.

- [201] C.C. Gonzaga. Path-following methods for linear programming. *SIAM Review*, 34:167–224, 1992.
- [202] C.C. Gonzaga and M.J. Todd. An  $O(\sqrt{n}l)$ -iteration large-step primal-dual affine algorithm for linear programming. *SIAM Journal on Optimization*, 2:349–359, 1992.
- [203] L. Gouveia. A  $2n$  constraint formulation for the capacitated minimal spanning tree problem. *Operations Research*, 43:130–141, 1995.
- [204] L. Gouveia. The asymmetric travelling salesman problem and a reformulation of the Miller-Tucker-Zemlin constraints. Centro de Investigação Operacional Working Paper 5/96, Universidade de Lisboa, 1996.
- [205] L. Gouveia. Using variable redefinition for computing spanning and Steiner trees with hop constraints. *INFORMS Journal on Computing*, 10:180–188, 1998.
- [206] S. C. Graves. Using Lagrangian techniques to solve hierarchical production planning problems. *Management Science*, 28:260–275, 1982.
- [207] H. J. Greenberg. RANDMOD: a system for randomizing modifications to an instance of a linear program. *ORSA Journal on Computing*, 3:173–175, 1991.
- [208] H. J. Greenberg. The use of the optimal partition in a linear programming solution for postoptimal analysis. *Operations Research Letters*, 15:179–185, 1994.
- [209] H. J. Greenberg. Consistency, redundancy, and implied equalities in linear systems. Mathematics department, University of Colorado at Denver, 1995.
- [210] H.J. Greenberg. *Design and Implementation of Optimization Software*. Sijthoff & Noordhoff, Alphen aan den Rijn, The Netherlands, 1978.
- [211] H.J. Greenberg. *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*. Kluwer Academic Publishers, Norwell, MA, 1993.
- [212] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, Berlin and New York, 1988.
- [213] M. Grötschel and O. Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33:243–259, 1985.

- [214] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs I: computation. Technical Report COC-94-09, Georgia Institute of Technology, 1994.
- [215] Z. Gu, G. L. Nemhauser, and M. W. P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs II: complexity. Technical Report COC-94-10, Georgia Institute of Technology, 1994.
- [216] M. Guignard and S. Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 39:215–228, 1987.
- [217] M. Guignard and K. Spielberg. Algorithms for exploiting the structure of the simple plant location problem. *Annals of Discrete Mathematics*, 1:247–271, 1977.
- [218] O. Güler, C. Roos, T. Terlaky, and J.-Ph. Vial. A survey of the implications of the behavior of the central path for the duality theory of linear programming. *Management Science*, 41:1922–1934, 1995.
- [219] G. Gunawardane, S. Hoff, and L. Schrage. Identification of special structure constraints in linear programming. *Mathematical Programming*, 21:90–97, 1981.
- [220] G. Hadley. *Linear Programming*. Addison-Wesley, Reading, MA, 1962.
- [221] L.A. Hall and R.J. Vanderbei. Two-thirds is sharp for affine scaling. *O.R. Letters*, 13:197–201, 1993.
- [222] N. Hall, J. Hershey, L. Dessler, and R. Stotts. A model for making project funding decisions at the National Cancer Institute. *Operations Research*, 40:1040–1052, 1992.
- [223] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5:1–28, 1973.
- [224] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.
- [225] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.
- [226] M. Held, P. Wolfe, and H. P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

- [227] D. den Hertog. *Interior Point Approach to Linear, Quadratic and Convex Programming Algorithms and Complexity*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1994.
- [228] D. den Hertog, C. Roos, and J.-Ph. Vial. A complexity reduction for the long-step path-following algorithm for linear programming. *SIAM Journal on Optimization*, 2:71–87, 1992.
- [229] D. Hilbert. Über die theorie der algebraischen formen. *Mathematische Annalen*, 36:473–534, 1890.
- [230] F.L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematics and Physics*, 20:224–230, 1941.
- [231] J. K. Ho and E. Loute. An advanced implementation of the Dantzig-Wolfe decomposition algorithm for linear programming. *Mathematical Programming*, 20:303–326, 1981.
- [232] A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 223–246. Princeton University Press, Princeton, NJ, 1956.
- [233] K. L. Hoffman and M. Padberg. LP-based combinatorial problem solving. *Annals of Operations Research*, 4:145–194, 1985.
- [234] K. L. Hoffman and M. Padberg. Solving airline crew scheduling by branch-and-cut. *Management Science*, 39:657–682, 1993.
- [235] K.L. Hoffman and M. Padberg. Improving LP-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3:121–134, 1991.
- [236] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Potomac, Maryland, 1978.
- [237] A.S. Householder. *Advanced Linear-Programming Computing Techniques*. Blaisdell, Waltham, MA, 1964.
- [238] J. K. Hurd and F. H. Murphy. Exploiting special structure in primal dual interior point methods. *ORSA Journal on Computing*, 1:70–83, 1989.
- [239] T. Ibaraki. Integer programming formulations of combinatorial optimization problems. *Discrete Mathematics*, 16:39–52, 1976.

- [240] E. Ignall and L. Schrage. Applications of the branch and bound technique to some flow shop scheduling problems. *Operations Research*, 13:400–412, 1965.
- [241] R.H.F. Jackson, P.T. Boggs, S.G. Nash, and S. Powell. Guidelines for reporting results of computational experiments. Report of the ad hoc committee. *Mathematical Programming*, 49:413–425, 1991.
- [242] B. Jansen, C. Roos, and T. Terlaky. A polynomial primal-dual Dikin-type algorithm for linear programming. *Mathematics of Operations Research*, 21:341–353, 1996.
- [243] R. G. Jeroslow, R. K. Martin, R. L. Rardin, and J. Wang. Gainfree Leontief flow substitution problems. *Mathematical Programming*, 57:375–414, 1992.
- [244] R.G. Jeroslow. Cutting-plane theory: Algebraic methods. *Discrete Mathematics*, 23:121–150, 1978.
- [245] R.G. Jeroslow. Some basis theorems for integral monoids. *Mathematics of Operations Research*, 3:145–154, 1978.
- [246] R.G. Jeroslow. An introduction to the theory of cutting-planes. *Annals of Discrete Mathematics*, 5:71–95, 1979.
- [247] R.G. Jeroslow. Minimal inequalities. *Mathematical Programming*, 17:1–15, 1979.
- [248] R.G. Jeroslow. *Logic-Based Decision Support Mixed Integer Model Formulation*. North-Holland, Amsterdam, 1989.
- [249] R.G. Jeroslow and J.K. Lowe. Modeling with integer variables. *Mathematical Programming Study* 22, 22:167–184, 1984.
- [250] R.G. Jeroslow and J.K. Lowe. Experimental results on the new techniques for integer programming formulations. *Journal of the Operational Research Society*, 36:393–403, 1985.
- [251] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting algorithms in combinatorial optimization. Technical Report 94-24, Universität Heidelberg, 1994.
- [252] E.L. Johnson. Modeling and strong linear programs for mixed integer programming. In S.W. Wallace, editor, *Algorithms and Model Formulations in Mathematical Programming*, pages 1–41. NATO ASI Series 51, 1989.

- [253] E.L. Johnson and M. Padberg. Degree-two inequalities, clique facets and biperfect graphs. *Annals of Discrete Mathematics*, 16:169–187, 1982.
- [254] M. Johnson, A. A. Zoltners, and P. Sinha. An allocation model for catalog space planning. *Management Science*, 25:117–129, 1979.
- [255] H.-W. Jung, R.E. Marsten, and M.J. Saltzman. Numerical factorization methods for interior point algorithms. *ORSA Journal on Computing*, 6:84–105, 1994.
- [256] J.E. Kalan. Aspects of large-scale in-core linear programming. In *Proceedings of ACM Conference*, pages 304–313, Chicago, 1971. Association for Computing Machinery.
- [257] R. Kannan and A. Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8:499–507, 1979.
- [258] L.V. Kantorovich. Mathematical methods in the organization and planning of production. *Management Science*, 6:366–422, 1960.
- [259] H. Karloff. *Linear Programming*. Birkhäuser, Boston, MA, 1991.
- [260] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.
- [261] W. Karush. Minima of functions of several variables with inequalities as side conditions. M.S. thesis, department of mathematics, University of Chicago, 1939.
- [262] L.G. Khachian. A polynomial algorithm in linear programming (in English). *Soviet Mathematics Doklady*, 20:191–194, 1979.
- [263] L.G. Khachian. A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.
- [264] A. Khintchine. A quantitative formulation of Kronecker’s theory of approximation. *Izvestiya Akademii Nauk SSR Seriya Matematika Akad. Nauk. SSSR*, 12:113–122, 1948.
- [265] D. Kirby and H.P. Williams. Representing integral monoids by inequalities. Technical Report OR 62, University of Southampton, 1994.
- [266] V. Klee and G.J. Minty. How good is the simplex algorithm. In O. Shisha, editor, *Inequalities III*, pages 159–175. Academic, New York, 1972.

- [267] D. E. Knuth. *The Art of Computer Programming Volume 2 Seminumerical Algorithms*. Addison-Wesley, Reading, MA, second edition, 1981.
- [268] G.J. Koehler, A.B. Whinston, and G.P. Wright. *Optimization Over Leontief Substitution Systems*. Elsevier Publishing Company, Inc., The Netherlands and New York, 1975.
- [269] D. A. Kohler. Projections of convex polyhedral sets. Operational Research Center ORC 67-29, University of California, Berkeley, 1967.
- [270] M. Kojima, N. Megiddo, and S. Mizuno. A primal-dual infeasible-interior-point algorithm for linear programming. *Mathematical Programming*, 61:263–280, 1993.
- [271] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior point algorithm for linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming Interior-Point and Related Methods*, pages 91–103. Springer-Verlag, New York and Berlin, 1989.
- [272] T. C. Koopmans. Optimum utilization of the transportation system. *Econometrica*, 17:136–146, 1949.
- [273] H. W. Kuhn. Solvability and consistency for linear equations and inequalities. *The American Mathematical Monthly*, 63:217–232, 1956.
- [274] H.W. Kuhn and A.W. Tucker. Linear inequalities and related systems. In J. Neyman, editor, *Proceedings 2nd Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley, CA, 1951.
- [275] J.-L. Lagrange. *Théorie des Fonctions Analytiques*. Impr. de la République, Paris, France, 1797.
- [276] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.
- [277] L. S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, New York, 1970.
- [278] L. S. Lasdon and R. C. Terjung. An efficient algorithm for multi-item scheduling. *Operations Research*, 19:946–969, 1971.
- [279] L.S. Lasdon, J. Plummer, and G. Yu. Primal-dual and primal interior point algorithms for general nonlinear programs. *ORSA Journal on Computing*, 7:321–332, 1995.

- [280] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.
- [281] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman Computational Solutions for TSP Applications*. Wiley, Chichester, 1985.
- [282] E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 1966.
- [283] C. Lemke. The dual method of solving the linear programming problem. *Naval Research Logistics Quarterly*, pages 36–47, 1954.
- [284] J.K. Lenstra, A.H.G. Rinnooy Kan, and A. Schrijver. *History of Mathematical Programming*. Elsevier Science Publishing Company, New York, 1991.
- [285] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
- [286] J.M.Y. Leung and T.L. Magnanti. Valid inequalities and facets of the capacitated plant location problem. *Mathematical Programming*, 44:271–291, 1989.
- [287] J.M.Y. Leung, T.L. Magnanti, and R. Vachani. Facets and algorithms for capacitated lot sizing. *Mathematical Programming*, 45:331–359, 1989.
- [288] M.J. Liberatore and T. Miler. A hierachical production planning system. *Interfaces*, 15:1–11, 1985.
- [289] B.W. Lin and R.L. Rardin. Controlled experimental design for statistical comparison of integer programming algorithms. *Management Science*, 25:1258–1271, 1980.
- [290] J. D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, 11:972–989, 1963.
- [291] J. W.-H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions of Mathematical Software*, 11:141–153, 1985.
- [292] J.N.M. van Loon. Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research*, 8:283–288, 1981.
- [293] J. Lorie and L. Savage. Three problems in capital rationing. *Journal of Business*, 28:229–239, 1955.

- [294] L. Lovász and H.E. Scarf. The generalized basis reduction algorithm. *Mathematics of Operations Research*, 1:751–764, 1991.
- [295] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.
- [296] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1984.
- [297] I. J. Lustig. Feasibility issues in a primal-dual interior point method for linear programming. *Mathematical Programming*, 49:145–162, 1990.
- [298] I. J. Lustig. The influence of computer language on computational comparisons: An example from network optimization. *ORSA Journal on Computing*, 2:152–161, 1990.
- [299] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and its Applications*, 152:191–222, 1991.
- [300] I. J. Lustig, R. E. Marsten, and D. F. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM Journal on Optimization*, 2:435–449, 1992.
- [301] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Computational experience with a globally convergent primal-dual predictor-corrector algorithm for linear programming. *Mathematical Programming*, 66:123–135, 1994.
- [302] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Interior point methods for linear programming: Computational state of the art. *ORSA Journal on Computing*, 6:1–14, 1994.
- [303] I. J. Lustig, R. E. Marsten, and D. F. Shanno. The last word on interior point methods for linear programming-for now. *ORSA Journal on Computing*, 6:35–36, 1994.
- [304] I. J. Lustig and E. Rothberg. Gigaflops in linear programming. *Operations Research Letters*, 18:157–165, 1996.
- [305] T. L. Magnanti and R. T. Wong. Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29:464–484, 1981.
- [306] T.L. Magnanti. Combinatorial optimization and vehicle fleet planning: Perspectives and prospects. *Networks*, 11:179–214, 1981.

- [307] T.J. Van Roy and L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.
- [308] A. S. Manne. Programming and economic lot sizes. *Management Science*, 4:115–135, 1958.
- [309] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, 1957.
- [310] H.M. Markowitz. *Portfolio Selection, Efficient Diversification of Investments*. John Wiley & Sons, New York, 1957.
- [311] R. E. Marsten, M. J. Saltzman, D. F. Shanno, G.S Pierce, and J. F. Ballintijn. Implementation of a dual affine interior point algorithm for linear programming. *ORSA Journal on Computing*, 1:287–297, 1989.
- [312] R. E. Marsten, R. Subramanian, M. J. Saltzman, I. J. Lustig, and D.F. Shanno. Interior point methods for linear programming: Just call Newton, Lagrange, and Fiacco and McCormick! *Interfaces*, 20:105–116, 1990.
- [313] E. Martello and P. Toth. *Knapsack Problems Algorithms and Computer Implementations*. John Wiley & Sons, New York, 1990.
- [314] R. K. Martin. *Optimization of Integer Programs with Special Ordered Sets of Variables*. PhD thesis, University of Cincinnati, 1980.
- [315] R. K. Martin. Generating alternative mixed-integer programming models using variable redefinition. *Operations Research*, 35:820–831, 1987.
- [316] R. K. Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10:119–128, 1991.
- [317] R. K. Martin, R.L. Rardin, and B.A. Campbell. Polyhedral characterization of discrete dynamic programming. *Operations Research*, 38:127–138, 1990.
- [318] R. K. Martin and L. Schrage. Subset coefficient reduction cuts for 0/1 mixed-integer programming. *Operations Research*, 33:505–526, 1985.
- [319] R.K. Martin and L. Schrage. Constraint aggregation and coefficient reduction cuts for mixed-0/1 linear programming. Technical report, University of Chicago, 1986.
- [320] S.T. McCormick. Making sprase matrices sparser: Computational results. *Mathematical Programming*, 49:91–111, 1990.

- [321] D. McDaniel and M. Devine. A modified Benders' partitioning algorithm for mixed integer programming. *Management Science*, 24:312–319, 1977.
- [322] K. A. McShane, C. L. Monma, and D. F. Shanno. An implementation of a primal-dual interior point method for linear programming. *ORSA Journal on Computing*, 1:70–83, 1989.
- [323] N. Megiddo. A note on degeneracy in linear programming. *Mathematical Programming*, 35:365–367, 1986.
- [324] N. Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming Interior-Point and Related Methods*, pages 131–158. Springer-Verlag, New York and Berlin, 1989.
- [325] N. Megiddo. On finding primal- and dual-optimal bases. *ORSA Journal on Computing*, 3:63–65, 1991.
- [326] N. Megiddo and M. Shub. Boundary behavior of interior point algorithms in linear programming. *Mathematics of Operations Research*, 14:97–146, 1989.
- [327] A. Mehrotra and M.A. Trick. A column generation approach to graph coloring. Technical report, Carnegie Mellon University, 1993.
- [328] S. Mehrotra. On finding a vertex solution using interior point methods. *Linear Algebra and its Applications*, 152:233–253, 1991.
- [329] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.
- [330] S. Mehrotra. Quadratic convergence in a primal-dual method. *Mathematics of Operations Research*, 18:741–751, 1993.
- [331] S. Mehrotra and J. Sun. An algorithm for convex quadratic programming that requires  $O(n^{3.5}l)$  arithmetic operations. *Mathematics of Operations Research*, 15:342–363, 1990.
- [332] S. Menon. *Decomposition of Integer Programs with Application to Cutting Stock and Machine Allocation*. PhD thesis, University of Chicago, 1997.
- [333] R.R. Meyer. A theoretical and computational comparison of ‘equivalent’ mixed integer formulations. *Naval Research Logistics Quarterly*, 28:115–131, 1975.

- [334] R.R. Meyer. Integer and mixed-integer programming models: General properties. *Journal of Optimization Theory and Applications*, 28:191–206, 1981.
- [335] H. Minkowski. *Geometrie der Zahlen*. Teubner, Leipzig, 1896.
- [336] P.B. Mirchandani and R.L. Francis. *Discrete Location Theory*. John Wiley & Sons, Inc., New York, 1990.
- [337] L.G. Mitten. Branch-and-bound methods: General formulation and properties. *Operations Research*, 18:24–34, 1970.
- [338] S. Mizuno. Polynomiality of infeasible-interior-point algorithms for linear programming. *Mathematical Programming*, 67:109–119, 1994.
- [339] S. Mizuno, M. Kojima, and M.J. Todd. Infeasible-interior-point primal-dual potential-reduction algorithms for linear programming. *SIAM Journal on Optimization*, 5:52–67, 1995.
- [340] S. Mizuno, M.J. Todd, and Y. Ye. On adaptive-step primal-dual interior-point algorithms for linear programming. *Mathematics of Operations Research*, 18:964–981, 1993.
- [341] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part I: linear programming. *Mathematical Programming*, 44:27–41, 1989.
- [342] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part II: convex quadratic programming. *Mathematical Programming*, 44:43–66, 1989.
- [343] R. D. C. Monteiro, I. Adler, and M.G.C. Resende. A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension. *Mathematics of Operations Research*, 15:191–214, 1990.
- [344] R. D. C. Monteiro, T. Tsuchiya, and Y. Wang. A simplified global convergence proof of the affine scaling algorithm. *Annals of Operations Research*, 47:443–482, 1993.
- [345] R.D. C. Monterio, T. Tsuchiya, and Y. Wang. A simplified global convergence proof of the affine scaling algorithm. *Annals of Operations Research*, 15:443–482, 1990.
- [346] J.J. Moré and S.J. Wright. *Optimization Software Guide*. SIAM, Philadelphia, PA, 1993.

- [347] T. S. Motzkin. *Beitrage zur Theorie der Linearen Ungleichungen*. PhD thesis, University of Besel, Jerusalem, 1936.
- [348] B.A. Murtagh. *Advanced Linear Programming*. McGraw-Hill, New York, 1981.
- [349] R.M. Nauss and R.E. Markland. Theory and application of an optimizing procedure for lock box location analysis. *Management Science*, 27:855–865, 1981.
- [350] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.
- [351] G.L. Nemhauser. *Introduction to Dynamic Programming*. John Wiley and Sons, New York, 1966.
- [352] G.L. Nemhauser. The age of optimization: Solving large-scale real-world problems. *Operations Research*, 42:5–13, 1994.
- [353] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi. MINTO a Mixed INTeger Optimizer. *Operations Research Letters*, 15:47–58, 1994.
- [354] G.L. Nemhauser and P.H. Vance. Lifted cover facets of the 0-1 knapsack polytope with gub constraints. *Operations Research Letters*, 16:255–263, 1994.
- [355] G.L. Nemhauser and L.A. Wolsey. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Mathematical Programming*, 46:379–390, 1990.
- [356] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, Philadelphia, PA, 1994.
- [357] E.G. Ng and B.W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM Journal of Scientific Computing*, 14:1034–1056, 1993.
- [358] W. Orchard-Hays. *Advanced Linear-Programming Computing Techniques*. McGraw-Hill, New York, 1968.
- [359] J.B. Orlin. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Sloan School of Management 1615-84, MIT, 1984.
- [360] J.B. Orlin. On the simplex algorithm for networks and generalized networks. *Mathematical Programming Studies*, 24:166–178, 1985.

- [361] M. Padberg. *Linear Optimization and Extensions*. Wiley, Berlin, Germany, 1995.
- [362] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–7, 1987.
- [363] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.
- [364] M. W. Padberg, T.J. Van Roy, and L. A. Wolsey. Valid inequalities for fixed charge problems. *Operations Research*, 32:842–861, 1984.
- [365] M.W. Padberg. A note on the total unimodularity of matrices. *Discrete Mathematics*, 14:273–278, 1976.
- [366] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 17:199–215, 1983.
- [367] M.W. Padberg and L.A. Wolsey. Trees and cuts. *Annals of Discrete Mathematics*, 17:511–517, 1983.
- [368] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
- [369] R. G. Parker and R. L. Rardin. *Discrete Optimization*. Academic Press, Inc., San Diego, CA, 1988.
- [370] J.-C. Picard and M. Queyranne. A network flow solution of some nonlinear 0-1 programming problems and applications to graph theory. *INFOR*, 20:394–422, 1982.
- [371] C. J. Piper and A. A. Zoltners. Some easy postoptimality analysis for zero-one programming. *Management Science*, 22:759–765, 1976.
- [372] F.A. Potra. An infeasible-interior-point predictor-corrector algorithm for linear programming. *SIAM Journal on Optimization*, 6:19–32, 1996.
- [373] A. A. B. Pritsker, L. J. Waters, and P. M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–108, 1969.
- [374] W.R. Pulleyblank. Polyhedral combinatorics. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 312–345. Springer-Verlag, Berlin, Germany, 1983.

- [375] R.L. Rardin. Linear programming and generalizations of  $b^+$  Leontief flows. Technical report, Purdue University, 1989.
- [376] R.L. Rardin and U. Choe. Tighter relaxations of fixed charge network flow problems. Industrial and Systems Engineering J-79-18, Georgia Institute of Technology, 1979.
- [377] R.L. Rardin and L. A. Wolsey. Valid inequalities and projecting the multi-commodity extended formulation for uncapacitated fixed charge network flow problems. Technical report, Purdue University, 1990.
- [378] J.K. Reid. A sparsity exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Mathematical Programming*, 24:55–69, 1969.
- [379] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer Verlag Lecture Notes in Computer Science 840, Berlin, Germany, 1994.
- [380] J. Renegar. A polynomial-time algorithm based on Newton's method for linear programming. *Mathematical Programming*, 40:59–95, 1988.
- [381] J.M.W. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, 17:200–207, 1970.
- [382] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, N. J., 1970.
- [383] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization An Interior Point Approach*. John Wiley & Sons, Inc, New York, 1997.
- [384] C. Roos and J.-Ph. Vial. Long steps with the logarithmic penalty barrier function in linear programming. In J. Gabszewicz, J.-F. Richard, and L. A. Wolsey, editors, *Economic Decision-Making: Games, Economics and Optimization: Dedicated to Jacques H. Dréze*, pages 433–441. Elsevier Science Publisher B.V., Amsterdam, 1986.
- [385] C. Roos and J.-Ph. Vial. A polynomial method of approximate centers for linear programming. *Mathematical Programming*, 54:295–305, 1992.
- [386] D.J. Rose and R.E. Tarjan. Algorithmic aspects of vertex elimination on graphs. In *Proceedings of 7th Annual Symposium on the Theory of Computing*, pages 245–254, New York, 1975. Association for Computing Machinery.

- [387] J. B. Rosen. The gradient projection method for nonlinear programming, I. linear constraints. *Journal Society Industrial Applied Mathematics*, 8:181–217, 1960.
- [388] G.T. Ross and R.M. Soland. Modeling facility locations problems as generalized assignment problems. *Management Science*, 24:345–357, 1977.
- [389] D.M. Ryan and B.A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, Amsterdam, 1981.
- [390] N.V. Sahinidis and I.E. Grossmann. Reformulation of the multi-period MILP model for capacity expansion of chemical processes. *Operations Research*, 40:S127–S144, 1992.
- [391] R. Saigal. A simple proof of the primal affine scaling method. Department of Industrial and Operations Engineering 92-60, University of Michigan, 1992.
- [392] R. Saigal. On the primal-dual affine scaling method. *Opsearch*, 31:261–278, 1994.
- [393] R. Saigal. *Linear Programming A Modern Integrated Analysis*. Kluwer Academic Press, Norwell, MA, 1995.
- [394] H.M. Salkin and K. Mathur. *Foundations of Integer Programming*. Elsevier Science Publishers Co., Inc., New York, 1989.
- [395] M.J. Saltzman. Mixed integer linear programming survey. *OR/MS Today*, April 1994.
- [396] M. A. Saunders. Major Cholesky would feel proud. *ORSA Journal on Computing*, 6:23–27, 1994.
- [397] M.A. Saunders. The complexity of LU updating in the simplex method. In R.S. Anderssen and R.P. Brent, editors, *The Complexity of Computational Problem Solving*, pages 214–230. University Press, Queensland, 1976.
- [398] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
- [399] M.W.P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831–841, 1997.

- [400] L. Schrage. Implicit representation of variable upper bounds in linear programming. *Mathematical Programming Study*, 4:118–132, 1975.
- [401] L. Schrage. Implicit representation of generalized variable upper bounds in linear programming. *Mathematical Programming*, 14:11–20, 1978.
- [402] L. Schrage. *Optimization Modeling with LINDO*. Brooks/Cole, Pacific Grove, CA, fifth edition, 1997.
- [403] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.
- [404] P.D. Seymour. Decomposition of regular matroids. *Journal of Combinatorial Theory (B)*, 28:305–359, 1980.
- [405] J. F. Shapiro. *Mathematical Programming Structures and Algorithms*. John Wiley & Sons, New York, 1979.
- [406] J. F. Shapiro. A note on node aggregation and Benders' decomposition. *Mathematical Program*, 29:113–119, 1984.
- [407] H. Sherali and W. Adams. A hierarchy of relaxations and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3:411–430, 1990.
- [408] H.J.S. Smith. On systems of linear indeterminate equations and congruences. *Philosophical Transactions*, 151:293–326, 1861.
- [409] Gy. Sonnevend. An analytical centre for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In *Lecture Notes in Control and Information Sciences 84*, pages 866–876. Springer-Verlag, New York and Berlin, 1985.
- [410] K. Speilberg. Algorithms for the simple plant-location problem with some side conditions. *Operations Research*, 17:85–111, 1969.
- [411] J. Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1:397–405, 1967.
- [412] G.J. Stigler. The cost of subsistence. *Journal of Farm Economics*, 27:303–314, 1945.
- [413] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I*. Springer Verlag, New York, 1970.

- [414] R. E. Stone and C. A. Tovey. The simplex and projective scaling algorithms as iteratively reweighted least squares methods. *SIAM Review*, 33:220–237, 1991.
- [415] L.M. Suhl and U.H. Suhl. A fast LU update for linear programming. *Annals of Operations Research*, 43:33–47, 1993.
- [416] U. H. Suhl and L. M. Suhl. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, 2:325–335, 1990.
- [417] E. R. Swart. P=NP. Technical report, University of Guelph, 1986.
- [418] D. J. Sweeney and R. A. Murphy. A method of decomposition for integer programming. *Operations Research*, 27:1128–1141, 1979.
- [419] D. J. Sweeney and R. A. Murphy. Branch and bound methods for multi-item scheduling. *Operations Research*, 29:853–864, 1981.
- [420] R. A. Tapia, Y. Zhang, and Y. Ye. On the convergence of the iteration sequence in primal-dual interior-point methods. *Mathematical Programming*, 68:141–154, 1995.
- [421] É. Tardos and S. Plotkin. Improved dual network simplex. In *Proceedings of the First ACM-SIAM Symposium on discrete Algorithms*, pages 367–376, 1990.
- [422] J. Telgen. Identifying redundant constraints and implicit equalities in systems of linear constraints. *Management Science*, 29:1209–1222, 1983.
- [423] W.F. Tinney. Comments on using sparsity techniques for power system problems. Technical Report RAI 3-12-69, IBM, 1969.
- [424] M.J. Todd. Large-scale linear programming: Geometry, working bases and factorizations. *Mathematical Programming*, 26:1–20, 1983.
- [425] M.J. Todd. A Dantzig-Wolfe-like variant of Karmarkar's interior-point linear programming algorithm. *Operations Research*, 38:1006–1018, 1990.
- [426] M.J. Todd and B.P. Burrell. An extension to Karmarkar's algorithm for linear programming using dual variables. *Algorithmica*, 1:409–424, 1986.
- [427] M.J. Todd and Y. Ye. A centered projective algorithm for linear programming. *Mathematics of Operations Research*, 15:508–529, 1990.

- [428] J. Tomlin. Pivoting for size and sparsity in linear programming inversion routines. *Journal of the Institute of Mathematics and Its Applications*, 10:289–295, 1972.
- [429] J. Tomlin. On pricing and backward transformation in linear programming. *Mathematical Programming*, 6:42–47, 1974.
- [430] J. A. Tomlin. Branch and bound methods for integer and nonconvex programming. In J. Abadie, editor, *Integer and Nonlinear Programming*, pages 437–450. North-Holland, Amsterdam, 1970.
- [431] J. A. Tomlin. A note on comparing simplex and interior methods for linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming Interior-Point and Related Methods*, pages 91–103. Springer-Verlag, New York and Berlin, 1989.
- [432] J.A. Tomlin. On scaling linear prgramming problems. *Mathematical Programming Study*, 4:146–166, 1975.
- [433] J.A. Tomlin and J.S. Welch. Finding duplicate rows in a linear programming model. *Operations Research Letters*, 5:7–11, 1986.
- [434] T. Tsuchiya. Global convergence of the affine scaling methods for degenerate linear programming problems. *Mathematical Programming*, 52:377–404, 1991.
- [435] T. Tsuchiya and M. Muramatsu. Global convergence of a long-step affine scaling algorithm for degenerate linear programming problems. *Mathematical Programming*, 52:377–404, 1991.
- [436] A.W. Tucker. On directed graphs and integer programs. IBM mathematical research project, Princeton University, 1960.
- [437] P. M. Vaidya. An algorithm for linear programming which requires  $O((m+n)n^2 + (m+n)^{1.5}n)l$  arithmetic operations. *Mathematical Programming*, 47:175–201, 1990.
- [438] T.J. Van Roy and L. A. Wolsey. Valid inequalities and separation for uncapacitated fixed charge networks. *Operations Research Letters*, 4:105–112, 1985.
- [439] T.J. Van Roy and L. A. Wolsey. Valid inequalities for mixed 0-1 programs. *Discrete Applied Mathematics*, 14:199–213, 1986.
- [440] P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

- [441] R. J. Vanderbei, M. S. Meketon, and B. A. Freedman. A modification of Karmarkar's linear programming algorithm. *Algorithmica*, 1:395–407, 1986.
- [442] R.J. Vanderbei. Splitting dense columns in sparse linear systems. *Linear Algebra and Its Applications*, 152:107–117, 1991.
- [443] R.J. Vanderbei. *Linear Programming Foundations and Extensions*. Kluwer Academic Publishers, Boston/London/Dordrecht, 1996.
- [444] R.J. Vanderbei and J.C. Lagarias. I.I. Dikin's convergence result for the affine-scaling algorithm. *Contemporary Mathematics*, 114:109–119, 1990.
- [445] A.F. Veinott, Jr. Extreme points of Leontief substitution systems. *Linear Algebra and Its Applications*, 1:181–194, 1968.
- [446] H. M. Wagner and T. M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89–96, 1958.
- [447] V. Weispfenning. Parametric linear and quadratic optimization by elimination. Technical Report MIP-9404, Universität Passau, 1994.
- [448] H. Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533, 1935.
- [449] J.H. Wilkinson. Error analysis of direct methods of matrix inversion. *Journal of the ACM*, 8:281–330, 1961.
- [450] H. P. Williams. Fourier-Motzkin elimination extension to integer programming problems. *Journal of Combinatorial Theory (A)*, 21:118–123, 1976.
- [451] H. P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, New York, 1978.
- [452] H. P. Williams. A characterisation of all feasible solutions to an integer program. *Discrete Applied Mathematics*, 5:147–155, 1983.
- [453] H. P. Williams. Fourier's method of linear programming and its dual. *The American Mathematical Monthly*, 93:681–695, 1986.
- [454] H. P. Williams. An alternative form for the value function of an integer program. Faculty of Mathematical Studies OR16, University of Southampton, 1988.
- [455] H. P. Williams. The elimination of integer variables. *Journal of the Operational Research Society*, 43:387–393, 1992.

- [456] H.P. Williams. The reformulation of two mixed integer programming problems. *Mathematical Programming*, 14:325–331, 1978.
- [457] D. Wilson. What is processor cache worth? *Unix Review*, pages 33–42, 1997.
- [458] P. Wolfe. The composite simplex algorithm. *SIAM Review*, 7:42–54, 1965.
- [459] L. A. Wolsey. Facets and strong valid inequalities for integer programs. *Operations Research*, 24:367–372, 1976.
- [460] L. A. Wolsey. The b-hull of an integer program. *Discrete Applied Mathematics*, 3:193–201, 1981.
- [461] L. A. Wolsey. Integer programming duality: Price functions and sensitivity analysis. *Mathematical Programming*, 20:173–195, 1981.
- [462] L.A. Wolsey. Uncapacitated lot-sizing with start-up costs. *Operations Research*, 37:741–747, 1989.
- [463] R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.
- [464] R.T. Wong. Integer programming formulations of the traveling salesman problem. In *Proceedings of 1980 IEEE International Conference on Circuits and Computers*, 1980.
- [465] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. In *ACM Symposium of Theory of Computing*, May 1988.
- [466] Y. Ye. An extension of Karmarkar's algorithm and the trust region method for quadratic programming. In N. Megiddo, editor, *Progress in Mathematical Programming Interior-Point and Related Methods*, pages 49–63. Springer-Verlag, New York and Berlin, 1989.
- [467] Y. Ye. An  $O(n^3)$  potential reduction algorithm for linear programming. *Mathematical Programming*, 50:239–258, 1991.
- [468] Y. Ye. A potential reduction algorithm allowing column generation. *SIAM Journal on Optimization*, 2:7–20, 1992.
- [469] Y. Ye. *Interior Point Algorithms Theory and Analysis*. John Wiley & Sons, Inc, New York, 1997.
- [470] Y. Ye and M. Kojima. Recovering optimal dual solutions in Karmarkar's polynomial algorithm for linear programming. *Mathematical Programming*, 39:305–317, 1987.

- [471] Y. Ye, O. Güler, R. A. Tapia, and Y. Zhang. A quadratically convergent  $O(\sqrt{nl})$ -iteration algorithm for linear programming. *Mathematical Programming*, 59:151–162, 1993.
- [472] Y. Ye, M.J. Todd, and S. Mizuno. An  $\sqrt{nl}$ -iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.
- [473] E. Zemel. Lifting the facets of zero-one polytopes. *Mathematical Programming*, 15:268–277, 1978.
- [474] S. Zhang. On anti-cycling pivoting rules for the simplex method. *Operations Research Letters*, 10:189–192, 1981.
- [475] Y. Zhang. On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem. *SIAM Journal on Optimization*, 4:208–227, 1994.
- [476] Y. Zhang and R. A. Tapia. A superlinearly convergent polynomial primal-dual interior-point algorithm for linear programming. *SIAM Journal on Optimization*, 3:118–133, 1993.
- [477] Y. Zhang, R. A. Tapia, and J.E. Dennis. On the superlinear and quadratic convergence of primal-dual interior point linear programming algorithms. *SIAM Journal on Optimization*, 2:304–324, 1992.
- [478] Y. Zhang and D. Zhang. On polynomiality of the Mehrotra-type predictor-corrector interior-point algorithms. *Mathematical Programming*, 68:303–318, 1995.
- [479] A. A. Zoltners and P. Sinha. Integer programming models for sales resource allocation. *Management Science*, 26:242–260, 1980.

---

## AUTHOR INDEX

- Adams, W., 626-627, 630  
Adler, I., 63, 249, 254, 264, 291,  
293, 300-302, 306, 309, 514,  
540  
Afentakis, P.B., 17, 415-416  
Aho, A.V., 330  
Ahuja, R.K., 28, 511-512, 523, 682  
Anbil, R., 21, 332  
Andersen, E.D., 534  
Anderson, D.R., 28, 100, 179, 220,  
391  
Anderson, K.D., 534  
Anstreicher, K., 291  
Applegate, D., 556  
Armendáriz, E.P., 109  
Armstrong, R.D., 327  
Bachem, A., 56, 114  
Bahl, H.C., 364  
Balas, E., 53, 55, 544, 546, 556,  
558, 560, 590-591, 613, 615,  
626-627, 629-630  
Bala, M., xiv  
Balinski, M., 354, 363  
Ballintijn, J.F., 254  
Barany, I., 17, 416, 555, 578, 598  
Barnes, E.R., 249  
Barnhart, C., 332, 603, 612-613,  
630  
Bartels, R.H., 448  
Bausch, D.O., 507  
Bayer, D., 266, 300  
Bazaraa, M.S., 28-29, 198, 215,  
264, 267, 410, 523  
Beale, E.M.L., 169, 325, 329  
Bean, J.C., 11, 329  
Benders, J.F., 349  
Berge, C., 679  
Bilde, O., 425, 578  
Billups, S.C., 308  
Birge, J.R., 29  
Bixby, R.E., 178, 197, 204, 209,  
215, 239, 476, 493, 513, 534,  
538, 556  
Blair, C.E., 126-127, 136-137  
Bland, R.G., 176, 219  
Boggs, P.T., 300, 682  
Bollobás, B., 679  
Bondy, J.A., 679  
de Boor, C., 297-298  
Borse, J., xiv  
Bosch, R.A., 291  
Boyd, E.A., 617-618, 620, 630  
Brøndsted, A., 635  
Bradley, G.H., 124, 499  
Bradley, S.P., 80  
Bratley, P., 682  
Brearley, A.L., 534  
Brown, G.G., 499, 507  
Burrell, B.P., 243  
Cabay, S., 124  
Cacioppi, P., xiv, 171, 180, 500  
Camion, P., 483, 486  
Camm, J.D., 4, 28, 533  
Campbell, B.A., 514, 581-582, 590  
Campbell, J.F., 15, 329  
Carpenter, T.J., 300  
Ceria, S., 549, 556, 560, 626-627,  
629-630

- Chalmet, L.G., 511  
 Chang, S.F., 540  
 Charnes, A., 176  
 Chinneck, J.W., 538  
 Choe, U., 573, 578, 613, 630  
 Choi, I.C., 282  
 Cholesky, A., 462  
 Chorman, T.E., 4  
 Chou, T.J., 124  
 Chvátal, V., 29, 126-127, 162, 543,  
     549  
 Clark, F.E., 61  
 Claus, A., 579  
 Collins, G.E., 124  
 Conte, S.D., 297-298  
 Cook, S.A., 670  
 Cook, W., 334, 519, 522, 549, 556  
 Cordier, C., 549  
 Cornuéjols, G., 18, 354, 415, 556,  
     560, 626-627, 629-630  
 Cosares, S., 514  
 Crowder, H.P., 208, 411, 544, 546,  
     549, 556, 558, 563, 620  
 Cunningham, K., xiv  
 Cunningham, W.H., 493, 500, 504,  
     511, 513, 597  
 Curtis, A.R., 215  
 Dantzig, G.B., 5-6, 14, 29, 36, 81,  
     143, 151, 176-178, 201, 255,  
     369, 390, 511, 569  
 Day, A.C., 472  
 Degraeve, Z., xiv, 682  
 Demar, R.F., xv  
 Dennis, J.E., 309  
 Desrochers, M., 332  
 Desrosiers, J., 332, 603  
 Dessler, L., 11  
 Devine, M., 365  
 Dietrich, B.L., 542  
 Dikin, I.I., 249  
 Dines, L.L., 36  
 Diophantos of Alexandria, 103  
 Doherty, M.E., 354, 364  
 Doig, A.G., 320, 324  
 Domich, P.D., 114, 124, 300  
 Dongarra, J.J., 464, 479  
 Dowd, K., 465, 477, 479, 682  
 Dravnieks, E.W., 538  
 Duffin, R.J., 36  
 Duff, I.S., 215, 475  
 Dumas, Y., 332, 603  
 Dzielinski, B.P., 17, 416  
 Eaves, B.C., 36, 81  
 Edmonds, J., 504, 520, 673  
 Efroymson, M.A., 354  
 Eppen, G.D., xv, 28, 68, 318, 537,  
     583, 590, 613, 630  
 Erickson, R.E., 585, 590  
 Erisman, A.M., 475  
 Erlenkotter, D., 425  
 Escudero, L.F., 542  
 Evans, J.R., xv, 28, 515, 4  
 Everett, H., 407, 434  
 Fang, S., 28, 234, 306, 309  
 Ferris, M.C., 308  
 Fiacco, A.V., 261, 265, 291, 306  
 Fisher, M.L., 14, 18, 329, 354, 398,  
     412, 415, 425, 434  
 Flood, A.M., 569  
 Folven, G., xv  
 Ford, Jr., L.R., 510, 523  
 Forrest, J.J.H., 208, 211, 215, 309,  
     327, 331, 450  
 Foster, B.A., 332  
 Fourer, R., 28  
 Fourier, J.B.J., 36, 40  
 Fox, B., 682  
 Francis, R.L., 19, 511  
 Freedman, B.A., 249  
 Freund, R.M., 258  
 Frisch, K.R., 261  
 Frumkin, M.A., 114  
 Fulkerson, D.R., 14, 215, 510-511,  
     520, 523, 569

- Garcia, C.B., 266, 297  
Garey, M.R., 319, 659  
Garfinkle, R.S., 28  
Gass, S.I., 29  
von zur Gathen, J., 114, 333  
Gauthier, J.M., 327, 331  
Gavish, B., 17, 415-416  
Gay, D.M., 28  
Geoffrion, A.M., 19, 319, 364, 398,  
    404, 425, 434  
George, A., 465, 469, 473  
Ghouila-Houri, A., 486  
Gibson, R., 32  
Giles, F.R., 515, 524  
Gill, P.E., 211, 215, 272, 292, 295,  
    306  
Gleeson, J., 538  
Glover, F., 513  
Goffin, J. -L., 390  
Goldfarb, D., 207-208, 215, 219,  
    512  
Goldman, A.J., 62  
Golub, G.H., 448  
Gomory, R.E., 17, 126-127, 416,  
    486, 543, 558, 562  
Gondzio, J., 534, 540  
Gonzaga, C.C., 250, 258, 291,  
    306-307  
Gould, F.J., 28, 68, 318  
Gouveia, L., xv, 570, 583, 598  
Graves, G.W., 19, 364, 499  
Graves, S.C., 17, 416  
Greenberg, H.J., 63, 202, 204, 215,  
    306, 538, 682  
Grossmann, I.E., 600, 613, 630  
Grötschel, M., 28, 177, 219, 511,  
    524, 546, 660, 671  
Guignard, M., 354, 428, 561  
Gunawardane, G., 201  
Gustavson, F.G., 464, 479  
Gu, Z., 560  
Güler, O., 63, 306, 309  
Hadley, G., 29  
Hall, L.A., 249  
Hall, N., 11  
Hane, C.A., 603, 612-613, 630  
Hao, J., 512  
Harris, P.M.J., 208, 210, 215  
Hattingh, J.M., 208  
Hax, A.C., 80  
Heath, M.T., 473  
Held, M., 411, 415, 434  
Henry, Jr., Clay, 32  
Hershey, J., 11  
den Hertog, D., 28, 307, 309  
Hilbert, D., 131  
Hirst, J.P.H., 327, 331  
Hitchcock, F.L., 507  
Hoffman, A.J., 486  
Hoffman, K.L., 534-535, 542, 556  
Hoff, S., 201  
Holland, O., 511  
Honda, G.A.M., xv  
Hopcroft, J.E., 330  
Horowitz, E., 330  
Householder, A.S., 156  
Ho, J.K., 386  
Hultz, J., 513  
Hundley, D.R., 507  
Hurd, J.K., 305  
Ibaraki, T., 573  
Ignall, E., 428  
Jackson, R.H.F., 682  
Jaikumar, R., 14, 329, 412, 425  
Jansen, B., 293  
Jarvis, J.J., 28-29, 198, 215, 523  
Jeroslow, R.G., xv, 126-127, 131,  
    136-137, 514, 546, 555, 573,  
    586, 588, 590, 595, 613, 630,  
    668, 601  
Johnson, D.S., 319, 659  
Johnson, E.A., 620

- Johnson, E.L., 21, 332, 541, 544, 546, 549, 556, 558, 563, 603, 613, 630  
 Johnson, M., 329  
 Johnson, S.M., 14, 511, 569  
 Jung, H.W., 475, 479  
 Jünger, M., 555  
 Kalan, J.E., 471  
 Kannan, R., 114, 124  
 Kantorovich, L.V., 5, 507  
 Karel, C., 327  
 Karloff, H., 29  
 Karmarkar, N.K., 219, 225, 234, 238, 242, 249, 254, 258, 291, 300, 306, 540  
 Karmarkar, U., 17, 415-416  
 Karp, A., 464, 479  
 Karp, R.M., 415, 434  
 Karush, W., 264  
 Kernighan, B.W., 28  
 Kern, W., 56  
 Khachiyan, L.G., 177, 219  
 Khintchine, A., 333  
 Kim, S., 428, 561  
 Kirby, D., 136  
 Klee, V., 177  
 Klingman, D., 513  
 Kochler, D.A., 513  
 Koehler, G.J., 582  
 Kohler, D.A., 54  
 Kojima, M., 240, 277, 291-292, 300, 302, 304, 308  
 Koopmans, T.C., 5, 507  
 Kranich, E., 310  
 Krarup, J., 425, 578  
 Kruskall, J.B., 486  
 Kuhn, H.W., 264  
 Lagarias, J.C., 249, 266, 300  
 Lagrange, J.-L., 306, 434  
 Lam, T.P.L., 124  
 Land, A.H., 320, 324  
 Langevin, A., 15, 329  
 Lasdon, L.S., 17, 309, 329, 416, 425  
 Lawler, E.L., 15, 28, 319, 524  
 Lee, J., 328  
 Lemke, C., 191, 215  
 Lenstra, H.W., 333  
 Lenstra, J.K., 15, 6  
 Leontief, W., 5  
 Leung, J.M.Y., 554-555  
 Liberatore, M.J., 16  
 Lin, B.W., 682  
 Lippert, W., xv  
 Little, J.D.C., 327  
 Liu, J.W., 465, 469, 473  
 van Loon, J.N.M., 538  
 Lorie, J., 11, 329, 406  
 Loute, E., 386  
 Louveaux, F., 29  
 Lovász, L., 28, 177, 219, 338, 519, 524, 546, 626-627, 660, 671  
 Lowe, J.K., 573, 590, 595, 613, 630  
 Luenberger, D.G., 264, 267, 542  
 Lustig, I.J., 25, 282, 297, 299-300, 303-304, 306, 308-309, 469, 478-479, 534  
 Magnanti, T.L., 80, 361, 365, 511-512, 554-555, 579, 28, 523  
 Manne, A.S., 17, 416, 424  
 Marchand, H., 549  
 Markland, R.E., 18  
 Markowitz, H.M., 446-447, 462, 479  
 Marsten, R.E., 25, 254, 282, 297, 299-300, 304, 306, 308-309, 319, 469, 475, 479, 534  
 Martello, E., 12, 413  
 Martin, J.B., xv  
 Martin, P.R., xv  
 Martin, R.K., 514, 523, 537, 549, 553, 555, 560, 581-583, 586, 588, 590, 613, 630, 668  
 Mathur, K., 28

- McAdam, S.J., 109  
McCormick, G.P., 261, 265, 291,  
    306  
McCormick, S.T., 540  
McDaniel, D., 365  
McShane, K.A., 282, 308  
Megiddo, N., 176, 240, 249,  
    264–266, 277, 292, 300, 304,  
    308  
Mehrotra, A., 332  
Mehrotra, S., 266, 297, 299,  
    308–309  
Meketon, M.S., 249  
Menon, S., xiv, 21, 25  
Meyer, R.R., 573  
Miller, T., 16  
Minkowski, H., 89, 137, 647  
Minty, G.J., 177  
Mirchandani, P.B., 19  
Mitra, G., 534  
Mitten, L.G., 319  
Mizuno, S., 277, 291–292, 300, 302,  
    304, 308  
Monma, C.L., 282, 308, 585, 590  
Monteiro, R.D.C., 63, 249, 264,  
    291, 293, 301–302, 306, 309  
Moré, J.J., 25, 324, 562  
Motzkin, T.S., 36, 40, 54  
Mulvey, J.M., 300  
Muramatsu, M., 249  
Murphy, F.H., 305  
Murphy, R.A., xv, 329–330, 429,  
    434  
Murray, W., 211, 215, 272, 292,  
    295, 306  
Murtagh, B.A., 215, 479  
Murty, K.G., 327  
Murty, U.S.R., 679  
Nabar, S., xiv  
Nash, S.G., 682  
Nauss, R.M., 18  
Nemhauser, G.L., 6, 18, 25, 28,  
    332, 354, 415, 549, 553, 555,  
    558–560, 562, 581, 603, 613,  
    630, 639  
Nemirovskii, A., 309  
Nesterov, Y., 309  
Newton, I., 306  
Ng, E.G., 466, 475, 479  
Noon, C.E., 11, 329  
Orchard-Hays, W., 178, 215  
Orden, A., 176  
Orlin, J.B., 28, 511–512, 523, 682  
Padberg, M.W., 29, 486, 511,  
    534–535, 541–542, 544, 546,  
    549, 553, 556, 558, 563, 597,  
    620  
Papadimitriou, C.H., 28, 524  
Parker, R.G., 28  
Peyton, B.W., 466, 475, 479  
Picard, J.-C., 597  
Pierce, G.S., 254  
Piper, C.J., 434  
Plotkin, S., 512  
Plummer, J., 309  
Porta, F.A., 308  
Powell, J.H.W., xv  
Powell, S., 682  
Pritsker, A.A.B., 329  
Pulleyblank, W.R., 53, 133, 515,  
    524, 613, 630, 520  
Purdy, G., 114  
Putthenpura, S., 28, 234, 306, 309  
Queyranne, M., 597  
Raffensperger, F., xiv  
Rapp, S.H., 507  
Rardin, R.L., xv, 28, 514, 525, 573,  
    578, 581–582, 586, 588, 590,  
    613, 629–630, 668, 682  
Rarick, D., 202  
Rauri, A.S., 533  
Ray, T.L., 354  
Reid, J.K., 207–208, 215, 459, 475

- Reinelt, G., 15, 555  
 Renegar, J., 291  
 Resende, M.G.C., 249, 254, 293,  
     300, 309, 540  
 Rhys, J.M.W., 597  
 Ribière, G., 327, 331  
 Rinaldi, G., 556  
 Rinooy Kan, A.H.G., 6, 15  
 Robinson, L., xiv  
 Rockafellar, R.T., 425, 617, 635  
 Rogers, J.E., 300  
 Roos, C., 28, 63, 283, 291, 293,  
     306–307  
 Rosenthal, R.E., 507  
 Rosen, J.B., 221  
 Rose, D.J., 468  
 Ross, G.T., 15, 329  
 Rothberg, E., 309  
 Rutherford, T., 334, 549  
 Ryan, D.M., 332  
 Ryan, J., 538  
 Sahinidis, N.V., 600, 613, 630  
 Sahni, S., 330  
 Saigal, R., 28, 249, 293, 306  
 Salkin, H.M., 28  
 Salton, G.J., 11, 329  
 Saltzman, M.J., 239, 254, 282, 306,  
     324, 331, 475, 479  
 Sankaran, J., xiv  
 Saunders, M.A., 211, 215, 272,  
     292, 295, 306, 452, 462  
 Saunders, P.B., 511  
 Savage, L., 11, 329, 406  
 Savelsbergh, M.W.P., 25, 534, 560,  
     562, 603, 613, 630  
 Scarf, H.E., 328, 334, 338, 549  
 Schmidt, C.P., 28, 68, 318  
 Schrage, L., 8, xv, 28, 171, 180,  
     201, 346, 428, 462, 500, 506,  
     549, 553, 555, 560, 682  
 Schrijver, A., 6, 28, 133, 177, 219,  
     486, 515, 519, 524, 546,  
     626–627, 660, 665, 671  
 Seymour, P.D., 493  
 Shalleross, D., 334, 549  
 Shanno, D.F., 282, 25, 254, 282,  
     297, 299–300, 304, 306,  
     308–309, 469, 479, 534  
 Shapiro, J.F., 342, 364, 425  
 Sherali, H.D., 28–29, 198, 215, 264,  
     267, 410, 523, 626–627, 630  
 Shetty, C.M., 264, 267, 410  
 Shi, J., xiv  
 Shmoys, D.B., 15  
 Shub, M., 249  
 Sieveking, M., 114, 333  
 Sigismondi, G.C., 25, 562  
 Sinha, P., 15, 327, 329  
 Sinnathamby, S., xiv  
 Slotts, R., 11  
 Small, R.E., 325  
 Smith, H.J.S., 124  
 Soland, R.M., 15, 329  
 Solomon, M.M., 603  
 Sonnevend, Gy., 266  
 Soumis, F., 332, 603  
 Speilberg, K., 354  
 Steiglitz, K., 28, 524  
 Stein, J., 137  
 Stigler, G., 5  
 Stoer, J., 635  
 Stone, R.E., 255  
 Stutz, J., 513  
 Subramanian, R., 282, 306  
 Suhl, L.M., 453, 479  
 Suhl, U.H., 453, 479  
 Sun, J., 309  
 Swart, E.R., 579  
 Sweeney, D.J., 4, xv, 28, 100, 179,  
     220, 329–330, 391, 429, 434  
 Sweeney, D.W., 327  
 Tanga, R., 21, 332

- Tanner, K., xiv  
 Tapia, R.A., 309  
 Tardos, E., 512  
 Tarjan, R.E., 468  
 Telgen, J., 538  
 Terjung, R.C., 17, 329, 416  
 Terlaky, T., 28, 63, 293, 306  
 Thapa, M. N., 29  
 Thienel, S., 555  
 Tinney, W.F., 468  
 Tistaert, J., xiv  
 Todd, M.J., 201, 219, 243, 291,  
     307–308, 390  
 Tomlin, J.A., 538, 211, 213, 272,  
     292, 295, 306, 309, 325, 327,  
     329, 331, 446, 450, 542, 571  
 Toth, P., 12, 413  
 Tovey, C.A., 255  
 Trick, M.A., 332  
 Trotter, L.E., 114, 124  
 Tsubakitani, S., 533  
 Tsuchiya, T., 249  
 Tucker, A.W., 62, 264  
 Ullman, J.D., 330  
 Vachani, R., 555  
 Vaidya, P.M., 291  
 Van Roy, T.J., 17, 416, 511,  
     553–555, 578, 598, 629  
 Van Slyke, R.M., 201  
 Van Wassenhove, L.N., 425  
 Vance, P.H., 332, 558, 603,  
     612–613, 630  
 Vanderbei, R.J., 29, 249, 540  
 Veiga, G., 249, 254, 300, 540  
 Veinott, Jr., A.F., 513, 582, 585,  
     590  
 Vial, J.-Ph., 28, 63, 283, 291,  
     306–307, 390  
 Viswanath, M., xiv  
 Wagner, D.K., 538  
 Wagner, H.M., 420, 579  
 Wang, J., 514, 586, 588, 590, 668  
 Wang, Y., 249  
 Waters, L.J., 329  
 Wegryn, G. W., 4  
 Weispfenning, V., 45  
 Welch, J.S., 538  
 Weyl, H., 136–137  
 Whinston, A.B., 513, 582  
 Whitin, T.M., 420, 579  
 Whitney, H., 519  
 Wilkinson, J.H., 445  
 Williams, H.P., 28, 36, 81,  
     124–126, 136, 138, 527, 534  
 Williams, T.A., 28, 100, 179, 220,  
     391  
 Wilson, D., 478  
 Witzgall, C., 300, 635  
 Wolfe, P., 176, 205, 211, 215, 369,  
     390, 411  
 Wolfe, P.M., 329  
 Wolsey, L.A., 17, 28, 127, 137, 416,  
     511, 549, 553–555, 558–559,  
     578, 597–598, 600, 629, 639  
 Wong, R.T., 361, 365, 579  
 Wood, D.E., 319  
 Wright, G.P., 513, 582  
 Wright, M.H., 211, 215, 272, 292,  
     295, 306  
 Wright, S.J., 25, 324, 562  
 Yang, H., xiv  
 Yannakakis, M., 601  
 Ye, Y., 28, 240, 258, 291, 308–309,  
     390  
 Yoshise, A., 277, 291, 302  
 Yu, G., 309  
 Zangwill, W.I., 266, 297  
 Zemel, E., 558–559  
 Zhang, D., 300  
 Zhang, S., 176  
 Zhang, Y., 300, 308–309  
 Zions, S., 364  
 Zoltner, A.A., 15, 329, 434

---

## TOPIC INDEX

- absolute value constraint, 568  
acyclic graph, 680  
affine combination, 637  
affine hull, 638  
affine independence, 638  
affine subspace, 638  
affine transformation, 243  
affirmative action rule, 505  
aggregate constraint, 43  
Air Products and Chemicals, 14  
algorithm  
    back substitution, 444  
    Benders' decomposition, 353  
    branch-and-bound, 320  
    branch-and-cut, 556  
    branch-and-price, 604  
    Dantzig-Wolfe, 374  
    disjunctive cutting plane, 629  
    dual affine scaling, 251  
    dual barrier, 273  
    dual simplex, 194  
    ellipsoid, 219  
    Euclidean, 106  
    gradient projection, 221  
    Hermite normal form, 112  
    Karmarkar's Projective  
        Algorithm, 231  
    LU update, 453  
    minimum degree ordering, 469  
    modified primal barrier, 287  
    network simplex, 498  
    primal affine scaling, 246  
    primal barrier, 269  
    primal-dual barrier, 278  
    primal-dual, 198  
    projection of extended  
        formulation, 616  
    ranking solutions for Lagrangian  
        dual, 430  
    reduced basis, 337  
    revised simplex with LU update,  
        453  
    revised simplex, 159  
    row- $ijk$  Cholesky factorization,  
        465  
    row- $ikj$  Cholesky factorization,  
        472  
    send-and-split, 587  
    shortest route, 420  
    simplex, 150  
    sparse SAXPY gather, 474  
    sparse SAXPY scatter, 473  
    subgradient optimization, 411  
    Wagner-Whitin, 420  
allowable decrease, objective  
    function coefficient, 93, 189  
allowable decrease, right hand  
    side, 68, 185  
allowable increase, objective  
    function coefficient, 93, 189  
allowable increase, right hand side,  
    68, 185  
alphabet, 664  
alternative dual optima, 64, 68, 94  
alternative primal optima, 64, 70  
American Airlines, 3, 21  
American Olean Tile, 16  
Anchorage, Alaska, 508

arc, 481, 678  
 arithmetic mean, 214  
 arithmetic model of computation, 657  
 artificial variable, 80, 202  
 assignment, 507  
 auxiliary variable (see problem reformulation), 110, 565  
 back substitution, 444  
 bang-for-buck ratio, 12  
 barrier algorithm (see interior point algorithm), 261  
 barrier function, 262  
 barrier parameter, 262  
 Bartels-Golub update, 448  
 basic feasible solution, 144, 644  
 basic solution, 144  
 basic variable, 144, 644  
 basis of a lattice, 105  
 basis, 144, 644  
 Bellcore, 21  
 Benders' decomposition, 350, 369, 630  
 cut, 353  
 master program, 351  
 relaxed master, 351  
 subproblem, 352  
 bias, 658  
 big M, 15, 314, 532, 684  
 big O, 657  
 binary linear program, 7, 313  
 binding constraint, 6, 638  
 bipartite graph, 507  
 Bland anticycling rule, 176  
 blending problem, 7  
 block angular matrix, 23, 384  
 branch-and-bound, 320, 555  
 branch-and-cut, 555–556  
 branch-and-price, 602  
 branching, 319  
 BTRAN, 165, 194, 447  
 bump matrix, 452

C (programming language), 478  
 capacitated plant location, 18, 507  
 capital budgeting, 10, 406  
 Carathéodory theorem, 639  
 cash flow matching, 29  
 central path, 266  
 central trajectory, 266  
 certificate of proof, 665  
 Chicago, Illinois, 506, 508  
 Chinese remainder theorem, 138  
 Cholesky factorization, 541  
 bordering method, 465  
 column, 466  
 inner-product, 466  
 numeric, 462  
 outer-product, 466  
 row method, 465  
 row- $ijk$ , 465  
 row- $ikj$ , 472  
 submatrix, 466  
 symbolic, 466  
 Chvátal function, 126  
 Chvátal-Gomory (C-G) cut, 543  
 Chvátal-Gomory rounding, 543  
 Clark's proposition, 61  
 clique inequality, 541  
 clique, 468, 678  
 closed half-space, 638  
 closed walk, 677  
 co-NP, 666  
 coefficient reduction, 535  
 column generation, 373–374, 390, 567, 602  
 column space geometry, 255  
 Committee on Algorithms (COAL), 682  
 complementary slackness, 61, 157, 191, 406  
 complexity classes, 663  
 concave function, 316  
 cone, 638  
 congruence, 106

- conic combination, 636  
conic hull, 636  
conjunctive normal form, 667  
connected graph, 677  
conservation of flow constraint, 16  
constraint aggregation, 543  
constraint aggregation, coefficient reduction (CACR), 545  
constraint generation, 351  
constraint set, 6  
convex combination, 636  
convex hull relaxation, 399, 567  
convex hull, 636  
convex set, 636  
convexity row, 373  
coupling constraint, 23, 329  
coupling variable, 24  
CPLEX, 25, 204, 215, 239, 309, 469, 560, 562  
CPM, 506  
crashing, 202  
Cray computer, 473  
crew scheduling, 20, 331  
cut  
    1-*k*, 549  
    Benders', 353, 613  
    Chvátal-Gomory (C-G), 543  
    clique, 541  
    constraint aggregation,  
        coefficient reduction  
        (CACR), 545  
    deep, 615  
    definition, 542  
    dicut, 629  
    disjunctive, 626  
    Fenchel, 617  
    generalized flow cover, 553  
    Gomory, 544, 560  
    Lagrangian, 560  
    minimal cover, 546  
    mixed integer linear  
        programming, 549  
    mixed integer rounding, 549  
    pure integer linear programming, 542  
cutset, 510, 679  
cutting plane (see cut), 542  
cycle, 678  
cycling in simplex algorithm, 168  
dangling node, 324  
Dantzig pivot rule, 151, 177  
Dantzig-Wolfe decomposition, 369, 394  
    master program, 373  
    restricted master program, 373  
    subproblem, 373–374  
DAXPY, 473  
decision problem, 662  
decision variable, 6  
deep cut, 615  
degeneracy and network simplex, 499  
degeneracy cone, 170  
degenerate dual solution, 74, 94  
degenerate primal solution, 69, 153, 168, 190  
demand vertex, 482  
density of nonzero elements, 21  
Devex pricing, 208  
dicut cut, 629  
diet problem, 5  
dimension, 636  
Diophantine equation, 103  
directed cycle, 678  
directed graph, 678  
directed path, 678  
directed trail, 678  
directed walk, 678  
disjunctive program, 318, 590  
dot product, 651  
dual affine scaling algorithm, 251  
dual alternative optima, 64, 68, 94  
dual angular matrix, 24  
dual barrier algorithm, 273

- dual degenerate solution, 74, 94
- dual linear program, 58, 157, 191
- dual multiplier, 58
- dual polyhedron, 60
- dual price, 65
- dual recession cone, 60
- dual simplex algorithm, 194
- dual steepest edge, 208
- dual variable, 58, 157, 184
- duality gap, 269, 405
- duality theory for linear programming, 57
- duality
  - Fenchel, 617
  - Lagrangian, 394
  - linear programming, 57
  - strong, 58
  - weak, 57
  - Weyl-Minkowski, 136
- dynamic lot size
  - multiproduct, 15, 415, 487, 531
  - single product, 487, 554, 575, 579
- dynamic programming, 420, 581
- edge, 677
- elementary matrix, 163
- elimination graph, 468
- ellipsoid algorithm, 177, 219
- equilibration, 214
- eta matrix, 163, 206, 447
- Euclidean algorithm, 106
- Euclidean norm, 651
- Eulerian matrix, 485
- extended polyhedral
  - representation (see problem reformulation), 570
- extreme point, 60, 65, 154, 638, 642
- extreme ray, 53, 60, 638
- face, 640
- facet, 642
- fathoming, 321
- feasibility testing, 537
- Fenchel duality, 617
- fill-in, 446
- finite basis theorem, 89, 370, 638, 650
- fixed charge constraint, 16, 314, 550
- fixed cost, 15
- float time, 17
- FORCE\_VECTOR, 475
- forrest, 678
- Forrest-Tomlin update, 450
- FORTRAN, 478
- forward substitution, 447
- Fourier-Motzkin elimination (see projection), 40
- Fredholm alternative, 50
- free variable, 217
- FTRAN, 165, 194, 447
- fundamental theorem of linear algebra, 50
- fundamental theorem of linear programming, 90, 154, 644
- gain, 513
- gather, 473
- Gaussian elimination (see projection), 38
- gcd (see greatest common divisor), 106
- generalized assignment, 12, 411, 547, 592
- generalized flow cover cut, 553
- generalized network, 513
- generalized upper bound, 201, 329, 373
- generalized variable upper bound, 201
- geometric mean, 214
- Gomory cut, 544
- Gomory function, 126
- gradient projection algorithm, 221
- gradient, 220

- Gramm-Schmidt  
    orthogonalization, 652
- greatest common divisor, 106
- group (additive), 105
- Hadamard inequality, 660
- Hamiltonian cycle, 674
- Harris minimum ratio test, 210
- headless hyperarc, 514, 679
- Hermite normal form, 112
- Hilbert basis, 131, 517
- Homart Development, 11
- homotopy, 266
- Horn clause, 667
- Hunt Wesson Food, Inc., 19
- hyperarc, 514, 679
- hyperplane branching, 329, 332
- hyperplane, 638
- IBM, 25
- ILOG, 25
- inconsistent system, 50
- incumbent solution, 320
- inner point, 639
- inner product, 651
- intbib.bib, 310
- integer finite basis theorem, 131
- integer linear optimization, 3
- integer linear program, 7, 313
- integer monoid, 105
- integer polyhedron, 133, 484, 515
- integer program, 7, 313
- integrality gap, 321, 405
- integrality property, 404
- integrality property, simple plant  
    location, 415
- interior point algorithm  
    dual affine scaling, 251  
    dual barrier, 273  
    Karmarkar's projective  
        algorithm, 231
- modified primal barrier, 287
- primal affine scaling, 246
- primal barrier, 269
- primal-dual barrier, 278
- interior point bibliography, 310
- interior point, 639
- interval matrix, 487
- inverse projection, 370
- Euclidean algorithm, 110
- proof of finite basis theorem, 89
- proof of finite Hilbert basis, 129
- proof of integer finite basis  
    theorem, 131
- proof of mixed integer finite  
    basis theorem, 135
- sensitivity analysis, 93, 101
- solving a linear program, 90
- solving a single linear  
    congruence, 118
- solving a system of linear  
    congruences, 121
- solving integer linear equalities,  
    122
- solving integer linear  
    inequalities, 127
- use in preprocessing, 539
- irreducible inconsistent system  
(IIS), 78, 538
- Karmarkar's Projective Algorithm,  
    231
- Karp reducibility, 664
- Karush-Kuhn-Tucker optimality  
    conditions, 264
- Khintchine's theorem, 333
- knapsack problem, 12, 79, 406,  
    413, 544, 603
- kth heaviest subset, 666
- L1 cache, 477
- L2 cache, 477
- Lagrangian cut, 560
- Lagrangian decomposition, 428,  
    561
- Lagrangian dual, 394–395
- Lagrangian function, 396
- Lagrangian relaxation, 403

- Lajitas, Texas, 32, 506  
language recognition problem, 662  
lattice basis, 105  
lattice, 105, 334  
least squares, 220, 254, 462  
Leontief directed hypergraph, 514, 679  
Leontief matrix, 513, 582  
Leontief substitution flow, 588  
Leontief substitution system, 513  
lexicographic, 176  
LIFO node selection, 324  
lifting, 557
  - down-lifting, 559
  - sequential, 559
  - simultaneous, 559
  - up-lifting, 559
LINDO Systems, Inc., 25  
LINDO, 25, 65, 186, 204, 210, 213, 533, 537, 562  
lineality space, 638  
linear optimization, 3  
linear program, 6, 143  
linear programming relaxation, 320, 393  
linear subspace, 220  
locality of reference, 476  
lock box location, 17, 354, 375, 415, 527  
loop interchange, 476  
loop optimization, 476  
loop unrolling, 477  
loss, 513  
LU decomposition, 439  
LU update, 446  
mantissa, 658  
Markowitz pivot rule, 446  
Mathematica, 232, 269  
matrix inverse
  - Bartels-Golub update, 448
  - Forrest-Tomlin update, 450
  - Markowitz pivot rule, 446
product form, 162, 447  
rank one update, 156  
Saunders' update, 452  
Sherman-Morrison-Woodbury update, 156  
Suhl and Suhl update, 453  
matroid polyhedra, 519  
matroid, 519  
max flow-min cut theorem, 510  
maximum flow, 508  
minimal cover cut, 546  
minimal cover, 546  
minimal face, 484, 643  
minimal Hilbert basis, 131  
minimal proper face, 646  
minimum batch size, 314  
minimum cost network flow, 481, 574  
minimum degree ordering algorithm, 469  
minimum ratio test, 145, 199, 210  
minimum spanning tree, 523, 596, 678  
Minkowski's theorem, 89, 370, 647  
MINTO, 25  
MIPLIB, 681  
mixed integer finite basis theorem, 135, 566  
mixed integer linear program, 7, 313  
mixed integer rounding cut, 549  
model reformulation (see problem reformulation), 527  
modified primal barrier algorithm, 287  
monoid, 105  
MPSX/370-MIP/370, 327  
multicommodity minimum cost network flow, 515  
multiple choice constraint, 329  
multiple pricing, 205

- multiproduct dynamic lot size (see dynamic lot size), 15  
National Cancer Institute, 11  
negative side constraint, 551  
NETLIB, 310, 681  
network flow linear program, 20  
network flow, 481, 540  
network matrix, 491  
network simplex algorithm, 498  
Newton's method, 267, 305  
node-arc incidence matrix, 482  
nonbasic variable, 144  
nondeterministic polynomial, 665  
nonnegativity constraint, 6  
nonzero fill in, 446  
NP, 664  
NP-complete, 669  
NP-hard, 670  
OB1, 25, 282, 308, 469  
objective function, 6  
**one hundred percent rule** (see sensitivity analysis), 80  
**one-*k* cut**, 549  
optimal value function of objective function coefficient, 101  
optimal value function of right hand side, 74  
optimality conditions, 405  
optimality conditions, linear program, 62  
orthogonal subspace, 651  
OSL, 25, 210–211, 282, 309, 387, 562  
P, 663  
parametric analysis, 186  
parent constraint, 550  
Pareto optimal, 361  
partial pivoting, 444  
partial pricing, 205, 386  
partition, 63  
path following (see interior point algorithm), 266  
path, 677  
penalty calculation in branch-and-bound, 325  
PERT, 506  
Phase I, 201–202, 389  
Phase II, 202  
piecewise linear function, 314  
pivot element, 148, 440  
pivot row, 149  
pointed polyhedral cone, 638  
pointed polyhedron, 638  
polar cone, 51  
polyhedral combinatorics, 520  
polyhedral cone, 47, 638  
polyhedral monoid, 105  
polyhedron, 36, 638  
polynomial algorithm, 659  
polynomial reducibility, 664  
polytope, 638  
portfolio optimization, 462  
positive definite matrix, 462  
positive side constraint, 550  
potential function, 236  
pre-Leontief matrix, 513, 582  
pre-Leontief substitution system, 582  
predictor-corrector, 297  
preprocessing  
    coefficient reduction, 535  
    feasibility testing, 537  
    finding IIS, 537  
    probing, 541  
    projection and inverse projection, 539  
    redundancy, 537  
    rounding, 534  
    scaling, 213, 542  
    tightening bounds, 535  
    variable fixing, 541  
primal affine scaling algorithm, 246  
primal alternative optima, 64  
primal barrier algorithm, 269

- primal degenerate solution, 69, 499
- primal linear program, 58
- primal steepest edge, 206
- primal-dual algorithm, 198
- probing, 541
- problem reduction, 664
- problem reformulation
  - absolute value constraint, 566
  - disjunctive, 590
  - dynamic programming, 581
  - finite mixed integer basis, 567
  - Leontief flow, 579
  - multicommodity network flow, 573
  - preprocessing, 533
  - separation, 595
  - shortest route, 581
  - special ordered set, 571
  - traveling salesman, 568
- Procter & Gamble Co., 3, 364
- product form of inverse, 163, 447
- projected gradient, 220, 261–262
- projection cone, 53
- projection matrix, 653
- projection
  - Benders' decomposition, 349
  - finding an IIS, 538
  - of a gradient, 220, 262
  - of a point, 652
  - of a polyhedron, 36, 639
  - of a TDI system, 522
  - of a variable
    - in a linear congruence relation, 114
    - in integer linear equalities, 122
    - in integer linear inequalities, 124
    - in linear congruences, 121
    - in linear equalities (see Gaussian elimination), 36
- in linear inequalities (see Fourier-Motzkin elimination), 39
- of extended formulation, 613–614
- disjunctive, 626
- mixed integer finite basis theorem, 617
- multicommodity, 629
- proof of Clark's proposition, 61
- proof of Farkas' lemma, 50
- proof of partition, 63
- proof of strict complementary slackness, 62
- proof of strong duality, 58
- proof of Weyl's theorem, 46
- sensitivity analysis, 65, 80, 186
- solving a linear program, 47
  - use in preprocessing, 539
- projective transformation, 229
- proper face, 641
- pseudo cost, 327
- pseudo random number, 462
- purification, 239
- radix, 658
- RAM (see random access memory), 471
- RANDMOD, 682
- random access memory, 477
- random number generator, 682
- random variable, 462
- rank one update, 156, 448
- ray, 638
- recession cone, 60, 638
- recurrence, 475
- reduced basis algorithm, 337
- reduced basis, 336
- reduced cost, 27, 67, 95, 145, 185
- redundant constraint, 48, 644
- Reid update, 459
- relaxation, 319, 393
- restriction, 319, 393
- revised simplex algorithm, 159

- right hand side, 6
- round-off error, 209
- rounding, 534
- saddlepoint, 404
- satisfiability, 667
- Saunders' update, 452
- SAXPY, 473
- scaling, 213, 542
- scatter, 473
- Sears, 11
- segregated storage, 345, 366
- semi-group, 105
- send-and-split algorithm, 585
- sensitivity analysis, 27, 65
  - objective function coefficient
    - 100% rule, 101, 190
  - objective function coefficient
    - optimal value function, 101
  - objective function coefficient, 93, 188
- parametric analysis, 186
- right hand side 100% rule, 80, 187
- right hand side optimal value function, 74
- right hand side, 65, 184
- separating hyperplane theorem, 640
- separation (in branch-and-bound), 319
- separation algorithm, 511
- separation problem, 511, 546, 595
- set partitioning branching, 331
- set partitioning, 331, 604
- sharp representation, 573
- Sherman-Morrison-Woodbury
  - update, 156, 448
- short direction, 334
- short vector, 334
- shortest route, 420, 506, 581
- side constraint, 550
- simple lower bound, 198
- simple plant location, 18, 354, 375, 415, 427, 527, 554, 631
- simple upper bound, 198
- simplex algorithm, 6, 150
  - convergence, 153
  - cycling, 168
  - degeneracy, 168
  - Harris ratio test, 210
  - minimum ratio test, 145, 210
  - numerical tolerance, 209
  - Phase I, 201
  - Phase II, 202
  - pivot column selection, 205
  - pivoting, 147
  - product form of the inverse, 162
  - revised simplex, 154
  - scaling, 213
- simple upper bound, 198
- sparse LU update, 446
- simplex, 638
- sink hyperarc, 514, 679
- sink vertex, 482
- size of a linear program, 660
- size of a rational number, 660
- size of an integer, 660
- slack, 6, 638
- sliding objective function, 242
- SOS constraint (see special ordered set), 329
- source hyperarc, 514, 679
- source vertex, 482
- spanning tree, 596, 678
- sparse LU update, 446
- sparse matrix storage, 471
- sparse matrix, 21, 154, 446
- sparse SAXPY gather, 474
- sparse SAXPY scatter, 473
- special ordered set (type 1), 329, 373, 427
- special ordered set (type 2), 571
- special ordered set branching, 329
- special structure, 21

- spike column, 448
- spike row, 452
- stalling, 176, 505
- standard form, 6, 73, 143
- steepest ascent, 425
- steepest descent, 220
- steepest edge pricing, 206
- strict complementary slackness, 62
- strong duality, 58
- strongly feasible tree, 499
- strongly polynomial algorithm, 512, 659
- subdifferential, 425
- subgradient optimization, 411
- subgradient, 409
- subgraph, 678
- subspace, 651
- subtour breaking constraint, 14, 569
- Suhl and Suhl update, 453
- sum of infeasibility, 203
- supernode, 475
- supply vertex, 482
- symmetric dual problem, 59
- tableau, 149
- tailless hyperarc, 514, 679
- tangential approximation, 425
- test problem
  - dynamic lot sizing, 682
  - linear programming, 681
  - mixed integer linear, 681
  - traveling salesman, 681
  - vehicle routing, 682
- theorems of the alternative, 49
- tight constraint, 6, 638
- tight formulation, 321, 527, 565
- tight relaxation, 319
- tightening bounds, 535
- TITAN command in LINDO, 537
- tolerance, 209
- totally dual integer (TDI), 516
- totally unimodular matrix, 482
- trail, 677
- trajectory, 266
- transportation, 19, 507
- transshipment, 482
- traveling salesman, 13, 568
- tree, 678
- Turing machine, 661
- Turing reducibility, 664
- UMPIRE, 327, 331
- unbounded linear program, 48
- uncapacitated plant location (see simple plant location), 18
- undirected graph, 677
- unimodular column operation, 105
- unimodular matrix, 105, 482
- unimodular row operation, 105
- University of Wuppertal, 310
- upper Hessenberg matrix, 449
- valid inequality (see cut), 542
- variable fixing, 541
- variable redefinition (see problem reformulation), 565
- variable splitting, 540
- variable upper bound, 201
- variance-covariance matrix, 462
- vector processing machine, 473
- vertex, 481, 677–678
- vertex-arc incidence matrix, 482
- Wagner-Whitin property, 579
- walk, 677
- weak duality, 57, 403
- Weyl's theorem, 46, 136, 650
- WHIZARD, 272
- Wolfram Research, 232