

DETERMINISTIC OPERATIONS RESEARCH

Modeling and Solving
Optimization Problems

DAVID J. RADIA, Jr.

WILEY

WILEY

CONTENTS

PREFACE

1 INTRODUCTION TO OPERATIONS RESEARCH

1.1 WHAT IS DETERMINISTIC OPERATIONS RESEARCH?

1.2 INTRODUCTION TO OPTIMIZATION MODELING

1.3 COMMON CLASSES OF MATHEMATICAL PROGRAMS

1.4 ABOUT THIS BOOK
EXERCISES

2 LINEAR PROGRAMMING MODELING

2.1 RESOURCE ALLOCATION MODELS

2.2 WORK SCHEDULING MODELS

2.3 MODELS AND DATA

2.4 BLENDING MODELS

2.5 PRODUCTION PROCESS MODELS

2.6 MULTIPERIOD MODELS: WORK SCHEDULING AND INVENTORY

2.7 LINEARIZATION OF SPECIAL NONLINEAR MODELS

2.8 VARIOUS FORMS OF LINEAR
PROGRAMS

2.9 NETWORK MODELS
EXERCISES

3 INTEGER AND COMBINATORIAL MODELS

3.1 FIXED-CHARGE MODELS

3.2 SET COVERING MODELS

3.3 MODELS USING LOGICAL
CONSTRAINTS

3.4 COMBINATORIAL MODELS

3.5 SPORTS SCHEDULING AND AN
INTRODUCTION TO IP SOLUTION
TECHNIQUES

EXERCISES

4 REAL-WORLD OPERATIONS RESEARCH APPLICATIONS: AN INTRODUCTION 126

4.1 VEHICLE ROUTING PROBLEMS

4.2 FACILITY LOCATION AND NETWORK
DESIGN MODELS

4.3 APPLICATIONS IN THE AIRLINE
INDUSTRY
EXERCISES

5 INTRODUCTION TO ALGORITHM DESIGN

5.1 EXACT AND HEURISTIC ALGORITHMS

5.2 WHAT TO ASK WHEN DESIGNING
ALGORITHMS?

5.3 CONSTRUCTIVE VERSUS LOCAL
SEARCH ALGORITHMS

5.4 HOW GOOD IS OUR HEURISTIC
SOLUTION?

5.5 EXAMPLES OF CONSTRUCTIVE
METHODS

5.6 EXAMPLE OF A LOCAL SEARCH
METHOD

5.7 OTHER HEURISTIC METHODS

5.8 DESIGNING EXACT METHODS:
OPTIMALITY CONDITIONS
EXERCISES

6 IMPROVING SEARCH ALGORITHMS AND CONVEXITY

6.1 IMPROVING SEARCH AND OPTIMAL
SOLUTIONS

6.2 FINDING BETTER SOLUTIONS

6.3 CONVEXITY: WHEN DOES IMPROVING
SEARCH IMPLY GLOBAL OPTIMALITY?

6.4 FARKAS' LEMMA: WHEN CAN NO
IMPROVING FEASIBLE DIRECTION BE

FOUND?
EXERCISES

7 GEOMETRY AND ALGEBRA OF LINEAR PROGRAMS

7.1 GEOMETRY AND ALGEBRA OF
“CORNER POINTS”

7.2 FUNDAMENTAL THEOREM OF LINEAR
PROGRAMMING

7.3 LINEAR PROGRAMS IN CANONICAL
FORM
EXERCISES

8 SOLVING LINEAR PROGRAMS: SIMPLEX METHOD

8.1 SIMPLEX METHOD

8.2 MAKING THE SIMPLEX METHOD MORE
EFFICIENT

8.3 CONVERGENCE, DEGENERACY, AND
THE SIMPLEX METHOD

8.4 FINDING AN INITIAL SOLUTION: TWO-
PHASE METHOD

8.5 BOUNDED SIMPLEX METHOD

8.6 COMPUTATIONAL ISSUES

EXERCISES

9 LINEAR PROGRAMMING DUALITY

- 9.1 MOTIVATION: GENERATING BOUNDS**
- 9.2 DUAL LINEAR PROGRAM**
- 9.3 DUALITY THEOREMS**
- 9.4 ANOTHER INTERPRETATION OF THE SIMPLEX METHOD**
- 9.5 FARKAS' LEMMA REVISITED**
- 9.6 ECONOMIC INTERPRETATION OF THE DUAL**
- 9.7 ANOTHER DUALITY APPROACH: LAGRANGIAN DUALITY**
- EXERCISES**

10 SENSITIVITY ANALYSIS OF LINEAR PROGRAMS

- 10.1 GRAPHICAL SENSITIVITY ANALYSIS**
- 10.2 SENSITIVITY ANALYSIS CALCULATIONS**
- 10.3 USE OF SENSITIVITY ANALYSIS**
- 10.4 PARAMETRIC PROGRAMMING**
- EXERCISES**

11 ALGORITHMIC APPLICATIONS OF DUALITY

- 11.1 DUAL SIMPLEX METHOD**

11.2 TRANSPORTATION PROBLEM
11.3 COLUMN GENERATION
11.4 DANTZIG–WOLFE DECOMPOSITION
11.5 PRIMAL–DUAL INTERIOR POINT
METHOD
EXERCISES

12 NETWORK OPTIMIZATION ALGORITHMS

12.1 INTRODUCTION TO NETWORK
OPTIMIZATION
12.2 SHORTEST PATH PROBLEMS
12.3 MAXIMUM FLOW PROBLEMS
12.4 MINIMUM COST NETWORK FLOW
PROBLEMS
EXERCISES

13 INTRODUCTION TO INTEGER PROGRAMMING

13.1 BASIC DEFINITIONS AND
FORMULATIONS
13.2 RELAXATIONS AND BOUNDS
13.3 PREPROCESSING AND PROBING
13.4 WHEN ARE INTEGER PROGRAMS
“EASY?”
EXERCISES

14 SOLVING INTEGER PROGRAMS: EXACT METHODS

- 14.1 COMPLETE ENUMERATION
- 14.2 BRANCH-AND-BOUND METHODS
- 14.3 VALID INEQUALITIES AND CUTTING PLANES
- 14.4 GOMORY'S CUTTING PLANE ALGORITHM
- 14.5 VALID INEQUALITIES FOR 0–1 KNAPSACK CONSTRAINTS
- 14.6 BRANCH-AND-CUT ALGORITHMS
- 14.7 COMPUTATIONAL ISSUES
- EXERCISES

15 SOLVING INTEGER PROGRAMS: MODERN HEURISTIC TECHNIQUES

- 15.1 REVIEW OF LOCAL SEARCH METHODS: PROS AND CONS
- 15.2 SIMULATED ANNEALING
- 15.3 TABU SEARCH
- 15.4 GENETIC ALGORITHMS
- 15.5 GRASP ALGORITHMS
- EXERCISES

APPENDIX A BACKGROUND REVIEW

[A.1 BASIC NOTATION](#)

[A.2 GRAPH THEORY](#)

[A.3 LINEAR ALGEBRA](#)

[**REFERENCE**](#)

[**INDEX**](#)

DETERMINISTIC OPERATIONS RESEARCH

Models and Methods in Linear Optimization

DAVID J. RADER, JR.
Rose-Hulman Institute of Technology
Department of Mathematics
Terre Haute, IN



Copyright © 2010 by John Wiley & Sons, Inc. All rights reserved

Published by John Wiley & Sons, Inc., Hoboken, New Jersey

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Rader, David J., 1970-

Deterministic operations research : models and methods in linear optimization / David J. Rader, Jr.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-470-48451-7 (cloth)

1. Operations research—Mathematics. 2. Mathematical optimization. 3.

Linear programming. I. Title.

T57.6.R245 2010

519.7'2—dc22

2009054232

10 9 8 7 6 5 4 3 2 1

To Cetta, Megan, and
Abby

PREFACE

This book represents the deterministic operations research course that I have taught to mathematics, computer science, and engineering students over the past 10 years. It deals with the study of linear optimization (both continuous and discrete), emphasizing the modeling of real problems as linear optimization problems and the designing of algorithms to solve them, covering topics in linear programming, network optimization, and integer programming.

In this book, I emphasize three aspects of deterministic operations research:

1. Modeling real-world problems as linear optimization problems.
2. Designing algorithms (both heuristic and exact methods) to solve these problems.
3. Using mathematical theory to improve our understanding of the problem, to improve existing algorithms, and to design new algorithms.

Although these aspects are important for both researchers and practitioners of operations research, they are not always in the forefront of operations research textbooks. It is true that many books highlight optimization modeling and algorithms to solve these problems; however, very few, if any, explicitly discuss the algorithm design process used to solve problems. This book is intended to fill that gap.

My intended audience for this book is a junior/senior mathematics course in deterministic operations research. It will serve as both comprehensive text for such a course and a reference book for future exploration of the subjects. The book can also be used for operations research courses in engineering departments and for first-year graduate courses in linear optimization. It is not meant to be a reference book for researchers, but to be a book for people wanting to learn about linear optimization problems and techniques. A student using this book should have been exposed to (multivariable) calculus and linear algebra, whenever possible.

I introduce concepts and approaches by mixing in many examples to both demonstrate and motivate theoretical concepts and then proving these results. Advanced ideas are described in a common-sense style that illustrates the

core aspects of the idea in ways that are easy to understand, motivates the reader to understand how to think about the problem, not just what to think, and illustrates this using concrete examples. Each chapter of the book is designed to be the continuation of the “story” of how to both model and solve optimization problems by using the specific problems (linear and integer programs) as guides. This enables the reader (and instructors) to see how solution methods can be derived instead of just seeing the final product (the algorithms themselves).

I begin this book with an introduction to linear optimization modeling, incorporating many common modeling constructs through examples. In class I typically take about 20–25% of the term to discuss various models and the modeling process, so it is no surprise that the modeling chapters encompass that much of the book. I feel it is important for students to be able to translate a real-world situation into a mathematical language (here as an optimization model), be able to solve it using an optimization package, and be able to interpret the results. It is also important for students to see that there is a difference between the model and the data, and I try to emphasize this by discussing the general form of various constructs. I end my discussion of modeling by illustrating some common operations research problems and how they are solved. In addition, the exercises discuss various applications that have appeared in the research literature over the past 20 years.

Once the importance and usefulness of linear and integer programming models have been illustrated, I take a different approach to solution methods than most books. Typically, at an introductory level, solution methods for optimization problems are presented in the following style: (a) the problem is defined, (b) a solution technique is described and illustrated, and (c) the solution method is proven correct (it gives the correct answer) and various properties are studied. I illustrate how to design methods to solve these problems by first introducing the algorithm design paradigm through the use of heuristic methods for common integer programming models and then discussing general optimization approaches and showing how to use the properties of linear programs to specialize these general methods. This illustrates the algorithm design process.

I have found that the algorithm design process (which is key to deterministic operations research) can best be introduced through heuristic

methods. This allows us to concentrate on the design itself without having to worry about finding the optimal solution. Once we have designed a few simple algorithms we are ready to search for optimal solutions by first learning through an example what properties an optimal solution must have and then designing (and improving) an algorithm to satisfy them.

One challenge facing instructors is the lack of a formal background in the subject, especially for mathematics professors. Many may not be familiar with how algorithms are designed, the importance of modeling and proper model formulation, and how models and the algorithms that solve them interact. This book will help such instructors understand these ideas and present them to the students. The book emphasizes the thought process involved with modeling optimization problems and both the design and the improvement of algorithms to solve these problems. Each chapter builds upon the previous ones to illustrate these different aspects. For example, by the time the reader is formally introduced to the simplex method (the primary algorithm for solving linear programs), we will have discussed how this algorithm could have been developed, giving the reader “ownership” of the algorithm.

This book is organized around the three aspects of deterministic operations research that I want to emphasize. The first four chapters discuss an introduction to operations research and optimization modeling. Chapters 5 through 8 discuss algorithm design for (continuous linear) optimization problems, culminating in the formal discussion of the simplex method. This includes an introduction to continuous optimization algorithms, convexity, Farkas’ lemma, and the algebraic and geometric study of polyhedra. All of this culminates in the simplex method. Many will find it unusual to see the simplex method appear as late as Chapter 8 in this book. This is by design; I want to emphasize the importance of modeling first (and hence there are four chapters on models) and the algorithm design process next. This means that the simplex method, the end result of the design process, is important but not the sole focus of the book. In fact, because it is central to much of what we discuss in the book, it is symbolic that the simplex method is found in the middle chapter.

Chapters 9 through 11 discuss the duality theory of linear programs and the algorithmic and modeling ramifications of the theory. I also discuss

Lagrangian duality and briefly mention optimality conditions for general nonlinear programs. In Chapters 10 and 11, I cover uses of duality, including sensitivity analysis, parametric programming, and such algorithmic uses as dual simplex, column generation, Dantzig–Wolfe decomposition, and the primal–dual interior point algorithm. Each of these topics highlights how theoretical results can be used to improve how we solve problems computationally.

Chapters 12 through 15 continue our algorithmic design theme by discussing network optimization and integer programming methods. I have incorporated some topics not typically given in introductory books, such as various label-correcting algorithms for the shortest path problem (as opposed to just Dykstra’s method), the importance of preprocessing and probing in integer programming, cover inequalities for 0–1 knapsack constraints, lifting of valid inequalities, and branch-and-cut algorithms. In fact, the integer programming material in Chapter 14 introduces methods that are being implemented in software packages for integer programs. I finish the book with a discussion of modern metaheuristic techniques for discrete optimization problems. The methods form the backbone of many heuristic approaches to hard discrete problems.

Supplemental material will be key to the course, which will be placed on a dedicated web site <http://www.rose-hulman.edu/~rader/DOR> for the book. This includes the appropriate models (in various modeling languages) for each example in Chapters 2, 3, and 4, as well as Maple and Matlab content for many calculations used in the book. This will enable students and instructors to learn the concepts being introduced without needing to worry about software implementations. A collection of solutions to exercises is available to students; in addition, a full solutions manual will be made available for instructors.

The material from this book can be covered entirely over a two-semester sequence of courses. The core material of this book is contained in Chapters 5–9. Any course that includes the Simplex method (Chapter 8) needs Chapters 5–7 as background material. When I teach a course during one term using the material from this book, I have covered the following chapters over 40 class meetings (50 minutes each): Chapters 1, 2, 3, 5, 6, 7, 8, 9, 11.1–11.3, 13, and 14.1–14.5. In this course, I spend 8–10 lectures on material from

Chapters 2 and 3 (optimization modeling); this is due to my personal bias toward the importance of optimization modeling, and users of this book would not need to spend as much time on these topics. I have included a more detailed topics list on the web site.

I have added various topics to this book that are optional materials to cover in a class, but provide the reader with more advanced topics for future study. These include (but are not limited to) sections on Farkas' lemma (Sections 6.3 and 9.5), Lagrangian duality (Section 9.7), parametric programming (Section 10.4), Dantzig–Wolfe decomposition (Section 11.3), primal–dual interior point algorithm (Section 11.6), network optimization algorithms (Chapter 12), and valid inequalities for 0–1 linear constraints and branch-and-cut algorithms (Sections 14.5 and 14.6).

It is impossible to write a book without any help or guidance, and that is especially true of this book. I want to thank John Wiley & Sons for publishing this book and to acknowledge in particular my editors Susanne Steitz-Filler and Jacqueline Palmieri for their assistance and guidance throughout this process.

I also want to acknowledge the comments I have received from colleagues Susan Martonosi and Kevin Hutson and other anonymous reviewers regarding the topics presented and their order and from the 10 years' worth of students at Rose-Hulman Institute of Technology who have used early versions of this manuscript as their primary material and have pointed out errors and omissions. Any errors that remain are of course my own.

I especially want to acknowledge Diane Evans, Richard Forrester, Allen Holder, and Mike Spivey, who have used this book in their classes. They have each provided me with numerous suggestions and comments on how to improve the presentation of material. Our conversations regarding the material has been immeasurable, and I cannot thank them enough for their contributions.

My colleagues in the Mathematics Department at Rose-Hulman Institute of Technology have been very supportive during the time I wrote this book. I particularly want to thank S. Allen Broughton, Ralph P. Grimaldi, and Jeffery J. Leader for their support and advice.

I could not even begin to work on such a project without the lifelong support of my parents Mary and David Rader. They sacrificed plenty to allow

me the education that led to this project, and words cannot express my gratitude.

Last, but certainly not least, I want to thank my wife Concetta for her help with this project. Her love and support, as always, is greatly appreciated. I also want to thank our children Megan and Abigail for being patient with me when I needed to work instead of play, and for knowing when I needed to play instead of work.

DAVID J. RADER, JR.

*Terre Haute, Indiana
April 2010*

CHAPTER 1

INTRODUCTION TO OPERATIONS RESEARCH

1.1 WHAT IS DETERMINISTIC OPERATIONS RESEARCH?

Today, many “operations” problems are routinely solved throughout business and industry. These include determining work shifts for large departments, designing production facilities to maximize the throughput, allocating scarce resources (such as raw materials and labor) to meet demands at a minimum cost, or determining which investments to place available funds. Such problems are very different from the traditional problems that students of mathematics, science, and engineering often face, because there are no physical laws that can be used; for example, knowing how to use Newton’s second law or Ohm’s law will not help solve these industrial problems. To handle such problems, different **mathematical models** must be used.

Mathematical Model A mathematical model is a collection of variables and the relationships needed to describe important features of a given problem.

Until World War II, most businesses and industries did not worry about such operations problems. This was partly because no formal mathematical discipline directly handled these problems, and partly because there was no urgent need for them. During World War II, military planners began working with scientists and mathematicians in order to apply a scientific approach to the management of the war effort, in which they began to devise mathematical models to deal with such issues. After the war, others began to

look at these techniques for industry, which brought about the beginning of **Operations Research**.

Operations Research Operations research (OR) is the study of how to form mathematical models of complex science, engineering, industrial, and management problems and how to analyze them using mathematical techniques.

Operations research is the name most often used to describe the mathematical study of such problems. In business, it is also known as *Management Science* or *Operations Management*. Typically, OR is divided into two areas: *deterministic operations research*, where all parameters are fixed, and *stochastic operations research*, where we assume some of the problem parameters are random. This book deals exclusively with deterministic operations research.

There have been many successful and interesting applications of operations research techniques since the 1940s. In fact, operations research has been used to solve many diverse problems.

1. *Scheduling ACC Basketball*. Nemhauser and Trick [67] discussed how they used OR techniques to determine successful schedules for the Atlantic Coast Conference basketball season, which consists of a round-robin schedule among 10 teams stretching from Maryland to Florida.
2. *Designing Communication Networks*. Balakrishnan, Magnanti, and Mirchandani [4] look at the problem of designing survivable networks with high bandwidth in order to minimize costs. These problems have great importance due to the explosion of phone, cable, and video services.
3. *Scheduled Aircraft Maintenance*. Gopalan and Talluri [51] studied how to schedule aircraft maintenance when there are many operating restrictions in place, such as making sure the aircraft is at the correct maintenance location within a certain amount of time.
4. *Truck Dispatching for Oil Pickup*. Bixby and Lee [16] look at the vehicle routing and scheduling problem that arise from an oil company's desire to better utilize truck fleets and drivers.
5. *Component Assignment in Printed Circuit Board Manufacturing*. Hillier and Brandeau [58] consider the problem of determining the minimum cost

assignment of various components to insertion machines that are used in the automated assembly of printed circuit boards.

OR is much more than the formation of mathematical models of problems; it also deals with the solution of these models and the evaluation of their solutions with regard to the real-world problem. In deterministic OR models, we typically formulate the problem as an optimization problem, where we want to minimize or maximize a function (such as costs, profits, time, etc.) given that the solution must satisfy certain restrictions and requirements (demand met, scarce resources, etc.). Such optimization problems are often referred to as **Mathematical Programs**.

Mathematical Program A mathematical program or *Optimization Model* is a mathematical structure where problem choices are represented by **decision variables**. Using these decision variables, mathematical programs look to optimize some **objective function** of the decision variables subject to certain **constraints** that limit the possible choices (values of decision variables). It can be written in the most general form (assuming maximization problem)

$$\max \{f(\mathbf{x}) : \mathbf{x} \in S\},$$

where \mathbf{x} is a vector of decision variables, $f(\mathbf{x})$ is the objective function, and the set S is the set of values for the decision variables satisfying all of the constraints.

The three highlighted terms in the above definition, decision variables, objective function, and constraints, represent the fundamental concerns of any OR model. In fact, it is the objective function that makes a model an optimization model. This was fairly revolutionary in 1947, when George Dantzig used an objective function to determine which decisions were better than others. Before that, there was not much thought given in modeling toward objective functions; instead, basic ground rules given by those in authority were used to guide the search for the “best answer.”¹

But how do we create a mathematical model? That is the real question in any study of operations research. The whole modeling process involves a four-stage cycle:

1. *Problem Definition and Data Gathering.* Here is where the real work

begins. Actually knowing what problem we wish to model is more important than most people realize. If we have no idea what we want to determine, how do we know if we have a reasonable solution or not? Next, a modeler must observe the system and collect data to estimate any parameters of the model.

2. *Creating the Mathematical Model.* Once the problem is defined and data have been collected, it is now time to develop a model of the situation. This involves determining decision variables, constraints, and objectives that are needed in the problem. Sometimes it involves only determining which formula we need to use. Models can be either deterministic, as studied in this book, or stochastic, in which case our data are more important because we need them to verify assumptions on probability distributions, means, and so on.

3. *Solving the Mathematical Model.* This can have many different forms. If we have a deterministic optimization model, we try to find the values of the decision variables that give the best objective function value. Sometimes, however, this is not tractable. For example, in the infamous traveling salesperson problem, a person has to travel to many different locations and wants to determine the shortest route to visit them all before returning home. For many instances, this problem cannot be solved to optimality. In that case, when a problem is that difficult or intractable, operations researchers often try to develop **heuristics** that find a reasonable solution in a short amount of time. For stochastic problems, a “solution” is not always easy to find. In many instances, **simulation** is used to approximate a good solution. Here, we simulate the system many times by generating random numbers for each random variable and determine from these simulations which alternative is best.

4. *Using the Mathematical Model to Make Real-World Predictions.* Once we have a solution to a mathematical model, we must use that answer to solve the real-world problem. While it is often the case that the values of our decision variables are what we will use, that is not always the case. For example, if we solve a mathematical program and get fractional values, and we wanted an integral solution, we may want to “fudge” the values a little, perhaps by rounding up or down, to get integers. However, this is not always a good idea, because we may end up with a solution that does not

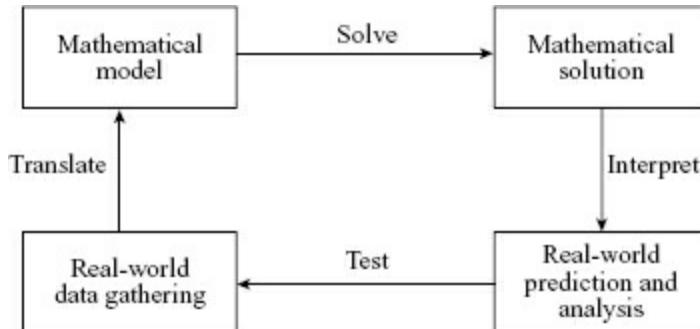
satisfy all the constraints.

Once we've gone through these four steps, we are done, right? Not quite. Often the model gives answers that we know cannot work in the real problem, which means the model is not quite right. In this case, we may have to start all over again. Mathematical modeling is a very cyclical process, as illustrated in [Figure 1.1](#).

Even when the model seems correct, we are often interested in scenarios where our parameters change. These "What If" scenarios are important, because often our data do not generate the absolutely correct parameters. Therefore, we want to know what small changes to our parameters do to our solutions.

Sensitivity Analysis Sensitivity analysis is the determination of the effect of small changes of parameter values on the solutions to a mathematical model.

FIGURE 1.1 Cyclical nature of mathematical modeling.



Sometimes this is very easy to do, and sometimes it is not. For one important model we'll study, sensitivity analysis is both straightforward and informative.

In this book we deal with three important parts of operations research: finding a mathematical model, solving it, and analyzing the results obtained and the solution technique itself. To do this, we first spend time on obtaining models, including some common models found in OR applications. This includes doing sensitivity analysis on certain models and learning what we can from the output of optimization software packages. We then begin a study of how to solve certain models by exploring approaches to algorithms for various mathematical problems. By learning how to derive such

algorithms, we will be able to handle problems where a solution technique is not apparent.

1.2 INTRODUCTION TO OPTIMIZATION MODELING

In order to understand the modeling process, we start with a simple example and show how some of the steps can be done. Since we will focus on optimization models, we'll begin with a two-variable problem, which enables us to solve and analyze the model graphically.

Model Formulation

Consider the following example.

■ EXAMPLE 1.1

Farmer Jones decides to supplement his income by baking and selling two types of cakes, chocolate and vanilla. Each chocolate cake sold gives a profit of \$3, and the profit on each vanilla cake sold is \$5. Each chocolate cake requires 20 minutes of baking time and uses 4 eggs and 4 pounds of flour, while each vanilla cake requires 40 minutes of baking time and uses 2 eggs and 5 pounds of flour. If Farmer Jones has available only 260 minutes of baking time, 32 eggs, and 40 pounds of flour, how many of each type of cake should be baked in order to maximize Farmer Jones' profit?

This is a classic problem type in operations research. Here, we want to choose the “best” solution (how many cakes to make) given a limited amount of resources (time, eggs, and flour). To model this problem, the first (and most important) step is to identify the **decision variables**. These are the variables that represent the decisions to be made. In Example 1.1, the decision variables correspond to determining how many chocolate cakes and vanilla cakes to bake. Here, we can define the following variables (of course, the names do not matter):

$$C \equiv \text{number of chocolate cakes to bake,}$$

$$V \equiv \text{number of vanilla cakes to bake.}$$

In this example, items such as baking time, amount of eggs and flour, and the restrictions on their use are all given to us; these quantities are often referred to as **input parameters**. These are fixed in any mathematical model we create.

Once we know the decision variables, we need to determine what restrictions and/or requirements govern their possible values. These formulations are known as the **constraints** of the problem and can be classified into two groups: (1) variable bounds and (2) general constraints. **Variable bounds** specify the values for which the decision variables have meaning. In Example 1.1, the variables C and V don't mean anything if their values are negative; hence, these variables are subjected to the most common variable bound, that of **nonnegativity**. Here, we would specify that

$$(1.1) \quad C, V \geq 0$$

in any solution to our model. Of course, in other models, the variables could also be either nonpositive (≤ 0) or unrestricted in sign (positive, negative, or zero). Note that these variables can (technically) have any value greater than or equal to zero.

Continuous Variable A variable that can have any fractional value is known as a continuous variable.

Discrete Variable If a variable is restricted to have only integer values, this variable would then be a discrete or integer variable.

General constraints specify all remaining restrictions, requirements, and other interactions that could limit the values of the decision variables. In Example 1.1, we are told that there are a limited number of eggs, pounds of flour, and baking time. Each one of these restrictions constitutes a constraint on the model. For the eggs, for example, the constraint would be of the form

$$(\# \text{ of eggs used to make the cakes}) \leq 32.$$

Note that this is an inequality since we do not necessarily need to use every available egg; we just cannot exceed our limit. We need to now determine how to find the number of eggs needed using the decision variables. Since we know that every chocolate cake requires 4 eggs and every vanilla cake uses 2 eggs, we find that the number of eggs used to make the cakes is 4 times the number of chocolate cakes made + 2 times the number of vanilla cakes made

$= 4C + 2V$. The constraint restricting the number of eggs used would then be

$$(1.2) \quad 4C + 2V \leq 32.$$

If we look at the amount of flour, we'd get a constraint of the form

$$(\text{pounds of flour used to make the cakes}) \leq 40.$$

Since each chocolate cake requires 4 pounds of flour and each vanilla cake requires 5 pounds of flour, the total amount of flower used is $4C + 5V$, giving the constraint

$$(1.3) \quad 4C + 5V \leq 40.$$

By this time, you can probably see that the constraint for the amount of baking time is

$$(1.4) \quad 20C + 40V \leq 260.$$

The last part of any mathematical program is the **objective function**, or the function of the decision variables that we wish to maximize or minimize. In Example 1.1, we are interested in maximizing the revenue. Similar to our construction of the constraints above, we need to come up with a function relating the decision variables to the revenue. Since each chocolate cake brings in \$3, and each vanilla cake brings in \$5, our objective is to maximize

$$(1.5) \quad 3C + 5V.$$

Example 1.1 can be written in the general form (1.7) by combining variable bounds (1.1), constraints (1.2), (1.3), and (1.4), and objective function (1.5), giving the following model:

$$\max \quad 3C + 5V$$

s.t.

$$4C + 2V \leq 32$$

$$4C + 5V \leq 40$$

$$20C + 40V \leq 260$$

$$(1.6) \quad C, V \geq 0.$$

This particular problem is an example of a **linear program**, because the objective function is linear and every constraint is a linear inequality (or equation).

Any mathematical program is a combination of the objective function, general constraints, and variable bounds. It is written in the following

general form:

$$\begin{aligned} & \text{max/min} && \text{(objective function)} \\ & \text{s.t.} && \text{(general constraints)} \\ (1.7) \quad & && \text{(variable bounds),} \end{aligned}$$

where “s.t.” stands for “subject to.”

Before we continue with our example, we should introduce some basic definitions.

Solution A solution to an optimization problem is a collection of values of the decision variables.

Feasible Solution A solution is feasible if it satisfies all constraints and bounds.

Feasible Region The feasible region of an optimization problem is the set of feasible solutions to the problem.

Value The value of a feasible solution is its objective function value.

Optimal Solution An optimal solution to an optimization problem is a feasible solution that is at least equal to all other feasible solutions. If our optimization problem is a maximization problem, then the optimal solution has value at least as large as all other feasible solutions. If it is a minimization problem, then the optimal solution has value no bigger than the value of any other feasible solution. The value of an optimal solution is known as the *optimal value*.

Unbounded Problem An optimization problem is unbounded if, for any feasible solution \mathbf{x} , there exists another feasible solution \mathbf{y} whose value is better than the value of \mathbf{x} . If the optimization problem is a maximization problem, then the value of \mathbf{y} is greater than the value of \mathbf{x} ; if it is a minimization problem, the value of \mathbf{y} is less than the value of \mathbf{x} . We often say that an unbounded optimization problem has *no finite optimal solution*.

Infeasible Problem A mathematical program is infeasible if there are no feasible solutions, that is, there does not exist any solution that satisfies all constraints and bounds.

Solving the Model

Now that we have a model for our problem, we need to solve it. If we were to plot the feasible region of problem (1.6), we'd have the shaded region given in [Figure 1.2](#). We are interested in finding the solution within this region that gives the largest value of $f(C, V) = 3C + 5V$. How do we find such a solution, if it even exists? It should be apparent that, because the feasible region is bounded and the boundary is well defined (because of the \leq constraints), there should be an optimal solution. One way to find this solution is to use an idea from calculus, namely *contour plots*, or curves of $f(C, V) = k$ for certain values of k . For example, [Figure 1.3](#) shows the feasible region, along with the lines $3C + 5V = 10$ (dots) and $3C + 5V = 20$ (dashes); the arrows indicate the direction in which to move the line $3C + 5V = k$ if we want to increase k . Note that each line intersects the feasible region, and hence there exists some solution in the feasible region that has an objective function value of 10, and another that has an objective function value of 20. The process should now be obvious: Find the largest value of k such that the line $3C + 5V = k$ intersects the feasible region.

Of course, we could just start guessing values of k until we find the right one, but shouldn't there be a better way? We can use the fact that all contours $3C + 5V = k$ are parallel. So, let a straightedge (like a ruler) represent the line $3C + 5V = k$ for some value of k such that the line intersects the feasible region. Here, perhaps an easy line to use would be $3C + 5V = 15$. Next, determine which direction to move the line so as to increase the value of k , and move the straightedge in that direction until our straightedge no longer intersects the feasible region. A solution that is both on the line and in the feasible region is then optimal. Note that, if we want to minimize our objective function, we'd move our straightedge in the opposite direction, but the same thinking would be involved.

FIGURE 1.2 Feasible region of problem (1.6).

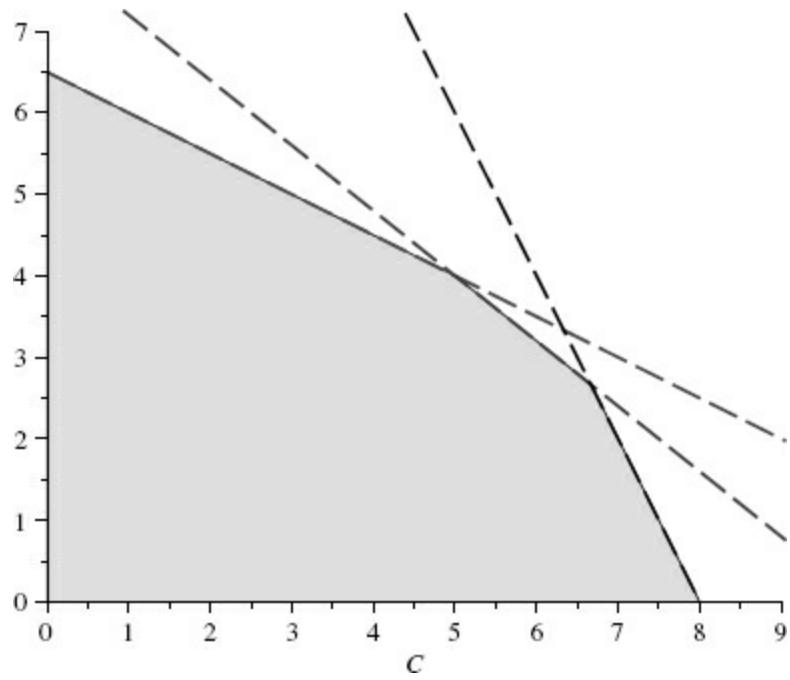


FIGURE 1.3 Contour lines for problem (1.6).

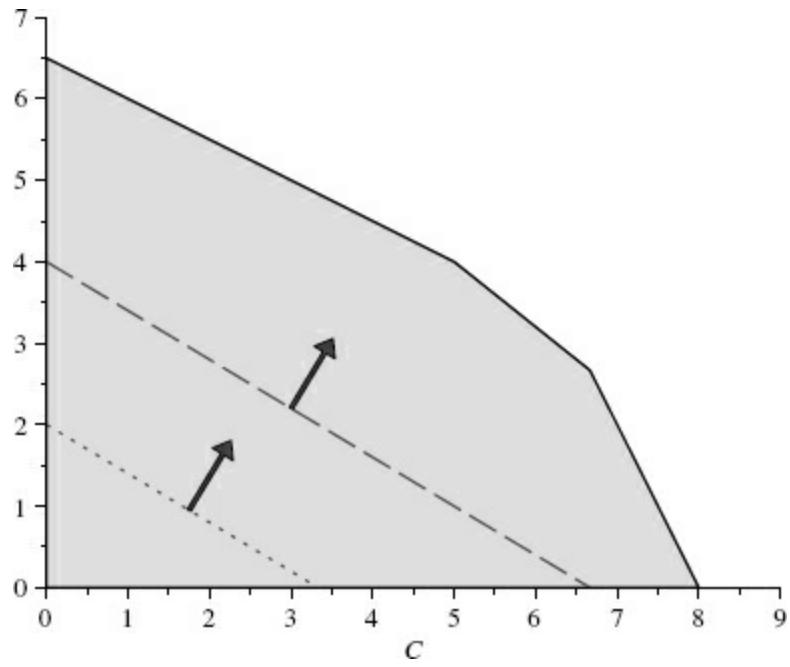
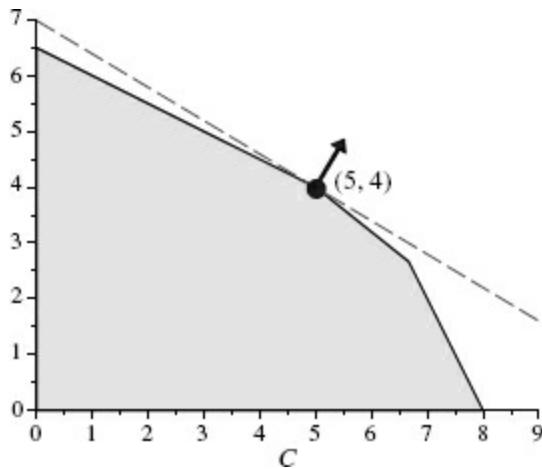


FIGURE 1.4 Optimal solution for problem (1.6) is at (5, 4).



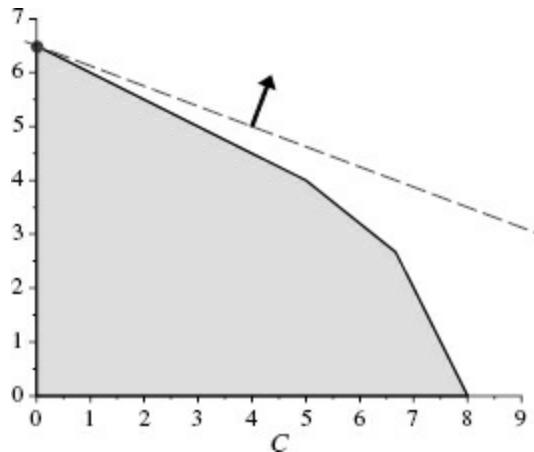
In problem (1.6), we first notice that increasing the value of k implies moving our straightedge in the direction of the arrows (which are the normal vectors for each line $3C + 5V = k$); [Figure 1.4](#) shows the results of drawing lines for the value $k = 35$ through the feasible solution $(5, 4)$, and any increase in k yields a line that no longer intersects the feasible region. Also, if we take any $k < 0$, we are again no longer intersecting the feasible region. Hence, every solution in the feasible region is on the line $3C + 5V = k$, for some k . Since we've in essence looked at all the solutions in the feasible region, if we started at $k = 0$, we could conclude that $C = 5$, $V = 4$ is an optimal solution to our mathematical program with value $3(5) + 5(4) = 35$.

This first example actually brings up a few questions that we may want to pay attention to, even though we won't be answering them right now. Our optimal solution is at a "corner point," or at the intersection of the two constraints $4C + 5V = 40$ and $20C + 40V = 260$. Will this always happen, or are there examples where no corner point is an optimal solution? Also, we found this optimal solution by increasing the value of k until our objective function contour line was no longer touching the feasible region. Is it possible for this contour line to intersect the feasible region for some k_1 , not intersect the feasible region for another $k_2 > k_1$, and then intersect the feasible region again for some $k_3 > k_2$? Why is this important, you may ask? If this can happen, then our method, if done by a computer, would stop at a solution that may be "locally" optimal but not "globally." These are issues that we want to keep in the back of our minds as we look at more examples.

What would happen if our profit margin for selling vanilla cakes changed from \$5 per unit to \$8? To examine this, plot the contours for $f(x, y) = 3C +$

$8V$. It's easy to see from [Figure 1.5](#) that the solution $(5, 4)$ is no longer optimal— $(0, 6.5)$ is. Can we really make 6.5 vanilla cakes? Probably not, so we may need to alter our model to specify that both V and C have integer value. This is an example of an **integer program**, which is a linear program where some (or all) the variables are restricted to integer values. However, restricting these variables to be integers alters our feasible region; only solutions with integer coordinates that satisfy the constraints are feasible, as shown in [Figure 1.6](#). How can we find the optimal solution now? The same way as before, by changing the contour lines. Doing so gives the optimal solution $(1, 6)$. Please note some major differences from the first problem: (1) the optimal solution is not at the intersection of two constraints anymore and (2) simply rounding the first solution (without the integer restriction) did not give the new optimal solution. This seems to suggest that solving problems with integer-restricted variables is more difficult in general than those problems where variables can take any value.

FIGURE 1.5 For an objective $z = 3C + 8V$, $(0, 6.5)$ is now optimal.



Sensitivity Analysis

As noted previously, it is useful to know how changes to the model parameters affect or do not affect the solution to our current model. Such sensitivity analysis provides insight into the nature of our solution and model, and is an important component of modeling. Let's see one example of this analysis for the Farmer Jones problem.

FIGURE 1.6 Feasible solutions (in dots) for integer-restricted problem.

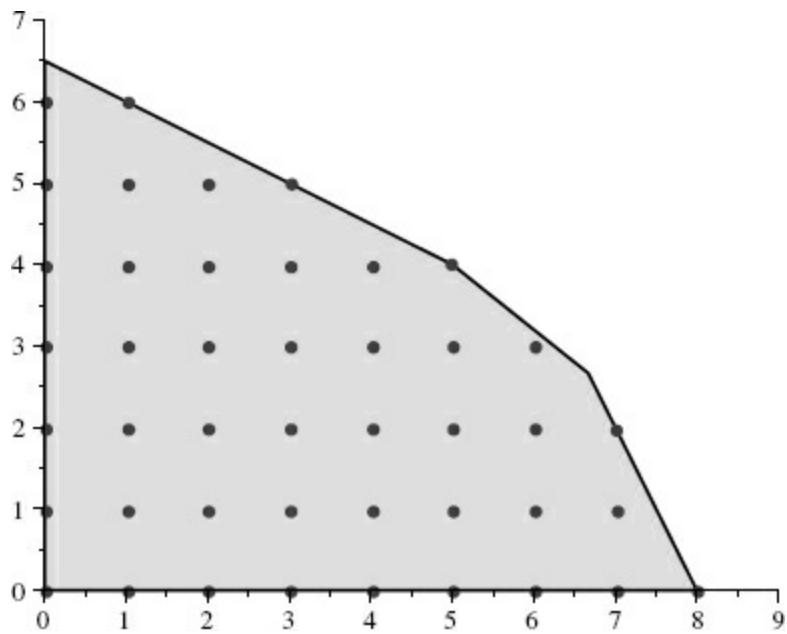
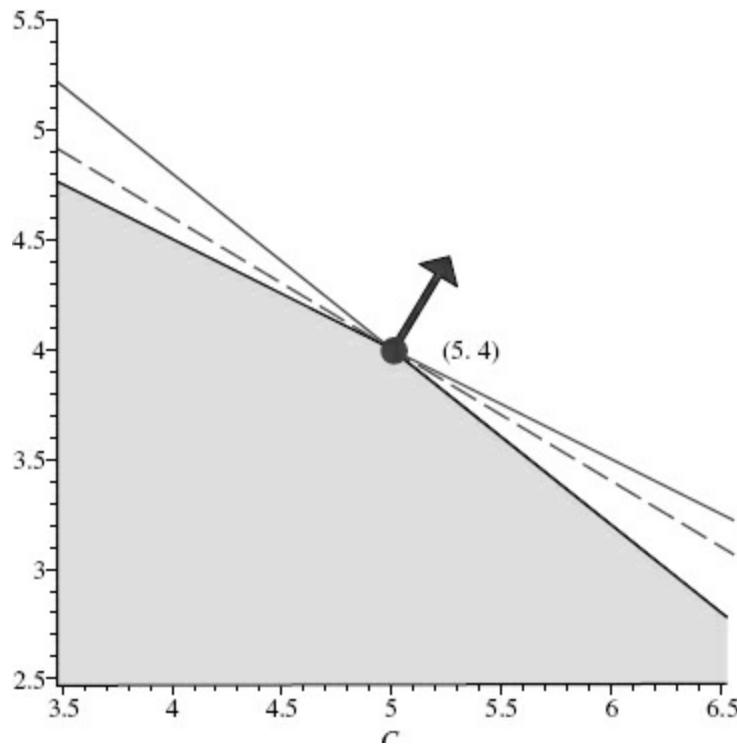


FIGURE 1.7 Sensitivity analysis for objective coefficient at optimal solution (5, 4).



We've already seen that if the profit margin for vanilla cakes changes from \$5 to \$8, our optimal solution changes. A natural question that Farmer Jones may ask is "For what profit margins on vanilla cakes will the current solution remain optimal?" Why is this a natural question? Farmer Jones may not want

to solve the linear program each time a value changes, so knowing what changes do not affect the optimal solution saves him time. To answer this question, refer back to how we solved the problem. In [Figure 1.4](#), we saw that the optimal solution occurs at the solution $(C, V) = (5, 4)$ because the contour $3C + 5V = k$ last intersects the feasible region here. If we were to change the profit margin of vanilla cakes from 5 to some value c_V , we are changing the slope of this line. In [Figure 1.7](#), we zoom in on the optimal solution $(5, 4)$ and show the current objective contour $3C + 5V = 35$ as the dashed line. It should be apparent that as long as the slope of our objective line stays between the slopes of the two constraints $4C + 5V = 40$ and $20C + 40V = 260$ (which intersect at $(5, 4)$), we have answered the question. Specifically, the solution $(C, V) = (5, 4)$ will remain optimal to our model as long as our profit margin c_V for vanilla cakes satisfies the following equation:

$$-\frac{4}{5} \leq -\frac{3}{c_V} \leq -\frac{1}{2},$$

where $-\frac{4}{5}$ is the slope of the line $4C + 5V = 40$, $-\frac{1}{2}$ is the slope of the line $20C + 40V = 260$, and $-\frac{3}{c_V}$ is the slope of the objective function. Solving this equation for c_V yields

$$\frac{15}{4} \leq c_V \leq 6.$$

Thus, if our profit margin increases by at most \$1 or decreases by no more than $\$(5 - \frac{15}{4}) = \1.25 it is best to make 5 chocolate cakes and 4 vanilla cakes.

1.3 COMMON CLASSES OF MATHEMATICAL PROGRAMS

In the Farmer Jones example we explored the concept of modeling using mathematical programs, including objective function and constraint generation. In doing so, we introduced two important classes of mathematical programs, linear programs and integer programs. In this section we will formally introduce these problems as well as introduce, through an example,

another common class.

Linear Programs

When the objective function is a linear function, and each constraint is either a linear inequality or linear equation, we say that the mathematical program (1.7) is a **Linear Programming Problem**, or a **Linear Program**. The general form of a linear program is

$$\begin{aligned} \text{max/min } & \sum_{j=1}^n c_j x_j \\ \text{s.t. } & \sum_{j=1}^n a_{ij} x_j \begin{cases} \leq \\ \geq \\ = \end{cases} b_i, \quad i \in \{1, \dots, m\} \\ & x_j \begin{cases} \geq 0 \\ \leq 0 \\ \text{unrestricted in sign} \end{cases}, \quad j \in \{1, \dots, n\}. \end{aligned}$$

For example, the optimization model (1.6) is a linear program. Linear programs are perhaps the most commonly used mathematical optimization models, having applications in many diverse areas such as production, inventory, finance, science, medicine, engineering, and pretty much any other discipline you can think of.

Of course, modeling a real-world problem as a linear program has a few implications. For example, having all constraints and the objective function as linear functions means that the contribution from each variable to the constraints and objective function must be proportional to the value of the variable. For example, in Example 1.1, we know that the profit (objective function contribution) from making 3 chocolate cakes (\$9) is exactly 3 times the profit of making 1 chocolate cake (\$3). This is also true for each constraint; in terms of eggs used, the number of eggs used in making 3 chocolate cakes (12) is 3 times the number of eggs used in making 1 chocolate cake (4). This idea is often referred to as the **Proportionality Assumption of Linear Programming**. Any model that violates this assumption cannot be modeled as a linear program. For example, if we can

make 3 chocolate cakes using only 10 eggs while we need 4 eggs for one cake, we cannot model this problem as a linear program.

Another assumption required in order to use linear programming to model situations addresses the total contribution all the variables make toward the objective function and the left-hand side of each constraint. We will make the assumption that these contributions are independent of all other variables. For example, in Example 1.1, the profit contribution for making C chocolate cakes is independent of how many vanilla cakes V are made; also, in the eggs constraint, the number of eggs used in making chocolate cakes is independent of how many vanilla cakes are made. Finally, the total contribution from all variables can be found by summing the contributions made from each variable. This assumption is often referred to as the **Additivity Assumption of Linear Programming**. If a mathematical model violates this assumption, it can still be formulated as a mathematical optimization problem, but this problem would be nonlinear, which is not as easy to solve as linear programs.

There are two other assumptions needed to model a real-world situation as a linear program. First, we assume that each variable can be infinitely divisible and still have meaning. For example, in Example 1.1, it makes sense to talk about 3.2162 chocolate cakes, if we are modeling average production over an extended period of time. This assumption is known as the **Divisibility Assumption**. Of course, not all models satisfy this assumption. For example, if our cake model is only for one week, we would want to have integer solutions. In this case, people will often ignore the assumption, get potentially fractional values for the variables, and then round the variables to the nearest integer. Unfortunately, this will not always yield a feasible solution.

The final assumption, and perhaps the most unrealistic, is the **Certainty Assumption**. Here we assume that every parameter (objective function coefficients, constraint coefficients, and right-hand side values) is known with certainty. While this rarely happens in the real world, we can handle this assumption by studying how changes in parameters affect (or do not affect) the optimal solution of the linear program. In this case, we'll see when it is acceptable to assume that parameters are fixed when they really are not.

Nonlinear Programs

One of the strengths of linear programs is that they are well understood mathematically and hence easy to solve. In fact, we show that it is possible to derive both algorithms for solving linear programs and criteria to easily identify when the current solution obtained is in fact optimal. Unfortunately, not all mathematical programs can be so easily solved. In many cases, either we cannot guarantee that the final solution is an optimal solution or, if we can get such a guarantee, the time to actually solve even moderately sized problems can be too long (say thousands of years!). However, one drawback of linear programs is that they can be too simple—in general, the world is not linear! Thus, other classes of models also often occur in practice.

One such model is a **nonlinear program**. In these problems, the additivity and proportionality assumptions of linear programming are often violated. In fact, you have probably been exposed to such problems early in your mathematical career, as the following example illustrates.

■ EXAMPLE 1.2

Suppose that we wish to enclose a rectangular yard using at most 100 feet of wire fencing. What is the largest area that can be formed?

You probably recall this example from calculus. If we let our decision variables be

$$l \equiv \text{length of the rectangular area},$$

$$w \equiv \text{width of the rectangular area},$$

then we can write our problem, in the form of (1.7), as

$$\max \quad lw$$

s.t.

$$2l + 2w \leq 100$$

$$l, w \geq 0.$$

Note that the objective violates the additivity assumption of linear programming, since the objective function contribution of each variable l and w depends on the other variable.

In general, we can write nonlinear programs in the form

$$\begin{aligned}
\text{max/min} \quad & f(x_1, \dots, x_n) \\
\text{s.t.} \quad & g_k(x_1, \dots, x_n) \quad \begin{cases} \leq \\ \geq \\ = \end{cases} \quad b_k, \quad k \in \{1, \dots, m\},
\end{aligned}$$

where f and each g_k are linear or nonlinear functions of the variables x_1, \dots, x_n . Their strength occurs in the fact that they are much more general than linear programs, and hence can model more complicated interactions between variables. However, nonlinear programs are more difficult to solve to optimality than linear programs, primarily because of the presence of local optimal solutions. Recall from calculus that these solutions are the “best” solution among those “nearby” feasible solutions, but may not be the optimal solution overall. Algorithms for nonlinear programs often stop when they have found such a local optimal solution, but they cannot comment on its global optimality.

Integer Programs

Perhaps the most common mathematical program used in business and industry is the **integer program**. Typically, these are linear programs where some (or all) of the variables are required to have integer values; this assumption violates the divisibility assumption of linear programming. Integer programs are often written in the form

$$\begin{aligned}
\text{max/min} \quad & \sum_{j=1}^n c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j \quad \begin{cases} \leq \\ \geq \\ = \end{cases} \quad b_i, \quad i \in \{1, \dots, m\} \\
& x_j \begin{cases} \geq 0 \\ \leq 0 \\ \text{unrestricted in sign} \end{cases}, \quad j \in \{1, \dots, n\} \\
& x_k \text{ integer}, \quad k \in S \subseteq \{1, \dots, n\}.
\end{aligned}$$

A relatively straightforward example of an integer program is given below.

■ EXAMPLE 1.3

Senior design students are trying to determine which person will take primary responsibility for the remaining tasks the team must complete. Below is a table of the amount of time each student would need to complete the corresponding task, in hours.

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Student 1	12	5	8	9	6	11
Student 2	14	8	7	11	10	5
Student 3	10	9	9	8	7	8
Student 4	11	8	10	10	9	10

If each student must do at least one but no more than two tasks, how should the tasks be divided so as to minimize the total amount of time required to finish all the tasks?

For this model, we want to identify which students get which tasks. One way to model this is to define our variables as follows:

$$x_{ij} = \begin{cases} 1, & \text{if student } i \text{ is assigned to task } j, \\ 0, & \text{otherwise.} \end{cases}$$

Each of these variables can have only the values 0 or 1, which automatically violates the divisibility assumption of linear programming. With these variables, we have the constraints that (1) each student can be assigned to either one or two jobs, and (2) each job must be assigned to at least one student. Constraint (1) implies that

$$1 \leq \sum_{j=1}^6 x_{ij} \leq 2, \quad i \in \{1, 2, 3, 4\}.$$

Constraint (2) is formed similarly:

$$\sum_{i=1}^4 x_{ij} \geq 1, \quad j \in \{1, 2, \dots, 6\}.$$

The objective function is to minimize $\sum_{i=1}^4 \sum_{j=1}^6 c_{ij} x_{ij}$, where c_{ij} is the time for student i to do job j . Our integer program is

$$\begin{aligned} \min \quad & 12x_{11} + 5x_{12} + 8x_{13} + 9x_{14} + 6x_{15} + 11x_{16} + \dots \\ & + 11x_{41} + 8x_{42} + 10x_{43} + 10x_{44} + 9x_{45} + 10x_{46} \end{aligned}$$

s.t.

$$\sum_{j=1}^6 x_{ij} \geq 1, \quad i \in \{1, 2, \dots, 4\}$$

$$\sum_{j=1}^6 x_{ij} \leq 2, \quad i \in \{1, 2, \dots, 4\}$$

$$\sum_{i=1}^4 x_{ij} \geq 1, \quad j \in \{1, 2, \dots, 6\}$$

$$x_{ij} \in \{0, 1\}, \quad i \in \{1, 2, \dots, 4\}, \\ j \in \{1, 2, \dots, 6\}.$$

If we can assign students to portions of tasks, then we could change the variable bounds $x_{ij} \in \{0, 1\}$ to

$$0 \leq x_{ij} \leq 1$$

for each $i \in \{1, 2, \dots, 4\}$ and $j \in \{1, 2, \dots, 6\}$ and thus have a linear program.

Integer programs are often used to model yes-or-no decisions, as the above example illustrates. It uses binary variables (those that can take only values 0 or 1) to do this, primarily because the logic of the model is easier to control. For example, a more natural approach in the above example is perhaps to let x_k be the task person k is assigned, so that $x_k \in \{1, 2, 3, 4\}$. However, by defining variables this way, it is difficult to write a constraint that says “no task can be assigned to more than one student.” Problems where all the variables are binary are often called **discrete or combinatorial optimization problems**.

Unfortunately, in general integer programs are very difficult to solve. Algorithms exist that, at least theoretically, will output the optimal solution when it finishes; however, the amount of time needed for the algorithm to finish may be more than the current age of the universe (independent of the speed of your computer)! Heuristic algorithms are often used for integer programs instead of exact algorithms. Heuristics typically stop in a rather

short amount of time, but are not guaranteed to find the global optimal solution. Such drawbacks usually are expected when working with integer programs. For combinatorial optimization problems, heuristics are often the best way to generate good solutions to the problem, since such problems are often computationally expensive to solve to optimality.

1.4 ABOUT THIS BOOK

To study and utilize the models an OR analyst would consider, it is best to look at three interconnected components. First, we must understand the logic and importance of optimization modeling. In fact, when presented with a real problem, we are not usually given a mathematical description of it. This is where the use of modeling techniques comes into play, and it is important for us to be comfortable with this part of the solution process. We study many such techniques in Chapters 2 and 3. We also explore in Chapter 4 some general classes of models that regularly appear in practice today.

Once we've developed a model for our problem, we have to know how to solve it. While many models we use are standard, and hence software has been developed to solve them, it is still important to know how to design algorithms, both exact and heuristic, that will solve our optimization model. To do so, in Chapter 5 we begin developing heuristic algorithms for common combinatorial optimization problems because they best illustrate the common aspects of algorithm design. After this study is done, in Chapters 6–8 we develop from first principles an algorithm that solves linear programs. We examine the notions of how to determine the structure of an optimal solution and how best to exploit this structure to develop our algorithm.

Once we have an algorithm that solves a linear program, in Chapter 9 we begin a study of the theory of linear programs that enables us to better understand and improve our algorithm and help design new algorithms. In Chapters 10 and 11, we exploit these new approaches to develop algorithms for special cases of linear programs.

Finally, in Chapters 12–15, we explore solution techniques for network-based optimization models and integer programs. These problems present unique challenges to us, both theoretically and algorithmically. During our study, we examine how the theory of linear programming plays a major role

in improving the basic algorithms for integer programs.

When we are done, we will have been introduced to the three main ideas of optimization modeling: model formulation, algorithm design, and algorithm improvement using the theory of the problem. We will have seen that all three areas are interconnected and really drive the development of new and improved models, algorithms, and theoretical results. And, we will have seen why deterministic operations research is such an important, useful, and exciting area to study. Let's begin!

EXERCISES

1.1 Solve the following linear programs by either identifying an optimal solution, indicating that the problem is unbounded, or showing the problem is infeasible.

(a)

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & 2x - y \leq 6 \\ & 2x + y \leq 10 \\ & x, y \geq 0. \end{aligned}$$

(b)

$$\begin{aligned} \max \quad & x + y \\ \text{s.t.} \quad & -2x + y \leq 0 \\ & x - 2y \leq 0 \\ & x + y \leq 9 \\ & x, y \geq 0. \end{aligned}$$

(c)

$$\begin{aligned} \max \quad & 25x_1 + 50x_2 \\ \text{s.t.} \quad & x_1 + 2x_2 \leq 300 \\ & 2x_1 + x_2 \leq 350 \\ & x_1 + x_2 \leq 200 \\ & x_1, x_2 \geq 0. \end{aligned}$$

(d)

$$\begin{aligned} \max \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 \geq 3 \\ & x_1 - x_2 \geq 1 \\ & x_1 + 2x_2 \leq 4 \\ & x_1, x_2 \geq 0. \end{aligned}$$

(e)

$$\begin{aligned} \max \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 \geq 3 \\ & x_1 - x_2 \geq 1 \\ & x_1 + 2x_2 \leq 2 \\ & x_1, x_2 \geq 0. \end{aligned}$$

1.2 For each problem in Exercise 1.1, determine the range of values for each objective coefficient so that, if that coefficient is the only parameter changed, the optimal solution originally found is still optimal.

1.3 Solve the integer program

$$\begin{aligned} \max \quad & 17x + 12y \\ \text{s.t.} \quad & 10x + 7y \leq 40 \\ & x + y \leq 5 \\ & x, y \geq 0, \text{ integer}. \end{aligned}$$

1.4 Solve the integer program

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & -4x + 6y \leq 9 \\ & 10x + 4y \leq 25 \\ & x, y \geq 0, \text{ integer}. \end{aligned}$$

¹Dantzig reminisces about this in Ref. [26].

CHAPTER 2

LINEAR PROGRAMMING MODELING

The importance of deterministic operations research in business and industry, among other places, is due to the enormous number of real-world applications that can be modeled as mathematical programs, and in particular as linear programs, integer programs, or network-based linear programs. In this chapter we explore some of the basic linear programming and network models used in deterministic operations research, and in the process, gain some experience in modeling general situations. The next chapter examines integer and combinatorial models. Finally, in Chapter 4 we explore some operations research models that commonly appear in practice and in the research literature.

2.1 RESOURCE ALLOCATION MODELS

A common linear programming model involves the allocation of scarce resources to optimize some function, typically the profit margin. We saw an example of this in Chapter 1, specifically the example of Farmer Jones baking cakes. In these models, the main issue is how to distribute resources among competing needs.

■ EXAMPLE 2.1

The Terre Haute Door Company (THDC) designs three types of steel doors: Standard, High Security, and Maximum Security. Each door requires different amounts of machine and labor time and has different profit margins;

this information is given in the following table.

	Machine 1		Machine 2		Profit Margin
	Hours	Manpower	Hours	Manpower	
Standard	3.5	5	4	6	\$35
High Security	6	8	5	7	\$45
Maximum Security	8	11	6	9	\$65

Each door must go through both machine 1 and machine 2 before it can be sold. Each worker is assigned to work on only one of the doors, which means they work on both machines. In addition, management has decided not to sell more Maximum Security doors than the combined total of Standard and High Security doors sold, in order to keep demand high for Standard and High Security doors. THDC has available to it only 120 hours per week on machine 1 and 100 hours on machine 2 before required maintenance, and 280 hours of manpower available per week. If we assume that we can sell every door that we make, how many of each door should be produced each week in order to maximize profits?

Typically, to begin the formulation of any mathematical model, we first define our variables. Since our decisions are the number of doors to produce of each type, our variables are

x_1 = number of Standard doors produced,

x_2 = number of High Security doors produced,

x_3 = number of Maximum Security doors produced.

Given these decision variables, our profit margin can be formulated as

$$\text{profit} = 35x_1 + 45x_2 + 60x_3.$$

We can now determine the number of hours used by machine 1, machine 2, and labor. For machine 1, we need 3.5 hours for each Standard door, 6 hours for each High Security door, and 8 hours for each Maximum Security door. This implies a total machine 1 time of $3.5x_1 + 6x_2 + 8x_3$. Since there are only 150 hours per week available on machine 1, we have the constraint

$$3.5x_1 + 6x_2 + 8x_3 \leq 120.$$

Similarly for machine 2, we have the constraint

$$4x_1 + 5x_2 + 6x_3 \leq 100.$$

Finally, the amount of labor time is calculated first for each machine and then

added together, since each worker will work at both machines. Thus, the total labor time used to produce x_1 Standard doors, x_2 High Security doors, and x_3 Maximum Security doors is

$$(5 + 6)x_1 + (8 + 7)x_2 + (11 + 9)x_3 = 11x_1 + 15x_2 + 20x_3.$$

Since only 280 hours of labor are available, this gives us the constraint

$$11x_1 + 15x_2 + 20x_3 \leq 280.$$

Finally, we want to restrict the number of Maximum Security doors sold to no more than the total combined number of Standard and High Security doors, leading to the constraint

$$x_3 \leq x_1 + x_2.$$

Note that this constraint has variables on both the left- and right-hand sides. This is perfectly acceptable, since simple algebra transforms the constraint into one that has all the variables on the left-hand side. Since we must produce a nonnegative amount of each type of door, we can combine all the above constraints and the profit calculations to produce the linear program given in Linear Program 2.1.

Linear Program 2.1 Resource Allocation Model for Example 2.1

$$\begin{aligned} \max \quad & 35x_1 + 45x_2 + 60x_3 \\ \text{s.t.} \quad & \\ & 3.5x_1 + 6x_2 + 8x_3 \leq 120 \quad (\text{machine 1}) \\ & 4x_1 + 5x_2 + 6x_3 \leq 100 \quad (\text{machine 2}) \\ & 11x_1 + 15x_2 + 20x_3 \leq 280 \quad (\text{labor}) \\ & x_3 \leq x_1 + x_2 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Solution The optimal solution is to produce 9.02336 Standard and 9.02336 Maximum Security doors for a profit of \$903.226. Of course, it may seem that this situation does not satisfy all the assumptions needed to formulate it as a linear program, namely that the variables should, perhaps, have only integer variables. If we add the constraint that the variables must be integers, the optimal solution is to produce 9 each of Standard and Maximum Security doors for an optimal profit of \$900. However, for the time being, we will

ignore this and assume that the number of doors can be fractional. This is not unrealistic if we are projecting over long periods of time.

It is also useful to examine the solution in terms of how it uses the available resources. Our optimal (fractional) solution uses all available labor hours, but more than 16 hours (out of the 120 hours) are still available on machine 1 and almost 10 hours are available on machine 2. These quantities can be calculated by taking the differences between the right-hand side and left-hand side of each constraint; for example, the slack amount of hours still available on machine 1 can be found by

$$s_{\text{machine 1}} = 120 - (3.5x_1 + 6x_2 + 8x_3).$$

This tells us that our labor availability is truly constraining us from generating greater profit, while we are completely underutilizing our machine usage.

Given any inequality constraint

$$\sum_j a_j x_j \leq b,$$

we can calculate the difference, or **slack**, s_l between the right- and left-hand sides of the inequality as

$$s_l = b - \sum_j a_j x_j.$$

Note that this quantity s_l must be nonnegative by definition. If we have the constraint

$$\sum_j a_j x_j \geq b,$$

we can calculate the difference, or **surplus**, s_g between the left- and right-hand sides of the inequality as

$$s_g = \sum_j a_j x_j - b.$$

If we have $s_l = 0$ ($s_g = 0$) at an optimal solution, then the constraint is called *binding*, which indicates that the solution is determined in part by that constraint. Improving our solution requires change in at least one of its binding constraints.

In addition, consider the management decision to restrict the number of Maximum Security doors sold. If we were to remove this constraint, our optimal solution would be to produce 14 Maximum Security doors for a profit of \$910. Thus, management's decision has the effect of diversifying the total production at a small reduction in overall profit.

This solution analysis raises some interesting modeling questions:

1. If we were to increase the number of labor hours available to us, by how much would our profit increase?
2. How much labor would we need to use before machine availability becomes an issue?
3. What effect would restricting the total production of each door have on the overall profit?

Let's consider the last question; the others will be left as an exercise (see Exercise 2.2). Suppose we insist that at most 8 of each door can be produced. If we add the constraints

$$x_k \leq 8$$

for $k = 1, 2, 3$, we see that our optimal solution is now to produce 8 each of Standard and Maximum Security doors and 2.1333 High Security doors, at a total profit of \$896. Thus, by reducing our optimal profit by roughly \$6 (which is less than 1%), we have a solution that produces all three types of doors. Note that, if the workers are relegated to working on only one type of door, our optimal solution would put some people out of work, but a good (but not optimal) solution keeps everyone employed. Such analysis would be beneficial to management when they use the results of this model to determine the appropriate course of action.

2.2 WORK SCHEDULING MODELS

Another example of resource allocation comes in the form of determining minimum cost solutions for satisfying various requirements. A common example of this is the design of work schedules to meet labor requirements.

■ EXAMPLE 2.2

A local manufacturing plant runs 24 hours a day, 7 days a week. During various times throughout the day, different numbers of workers are needed to run the various machines. Below are the minimum number of people needed to safely run the plant during the various times.

Times	Employees Needed
12AM–4AM	8
4AM–8AM	9
8AM–12PM	15
12PM–4PM	14
4PM–8PM	13
8PM–12AM	11

Each worker works two consecutive 4-hour periods. What is the minimum number of workers needed to safely run this plant?

Before we discuss a correct model for this problem, let's first look at an *incorrect* model that might at first seem correct. If we let our decision variables x_j be the number of employees working during the j th 4-hour period, $j = 1, \dots, 6$, starting with the 12 AM–4 AM shift, then it would appear reasonable that the number of employees needed would be the solution to the following linear program:

$$\begin{aligned} \min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\ \text{s.t.} \quad & x_1 \geq 8 \\ & x_2 \geq 9 \\ & x_3 \geq 15 \\ & x_4 \geq 14 \\ & x_5 \geq 11 \\ & x_6 \geq 13 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0. \end{aligned}$$

So, why is this incorrect? First, the sum $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$ in fact counts each person 2 times, not once. This is not a big deal, since we can divide the final answer by 2 and get the number needed. However, it is the

constraints that cause the biggest concern. Every variable, as listed, is interrelated, in that some people who work the first 4-hour period also work the second 4-hour period. Our constraints do not reflect this. Furthermore, these interrelations are *nonlinear* in nature, and thus, if we use these variables, a linear model cannot be used.

To model this linearly, the key is to realize that our decision variables are not how many people work during a given 4-hour period but *how many start at the beginning of each 4-hour period*. In this way, we can keep track of who is working when. Therefore, define

$$x_i = \begin{aligned} &\text{number of employees beginning their shift at} \\ &\text{the start of the } i\text{th 4-hour period, } i = 1, \dots, 6. \end{aligned}$$

For example, x_4 indicates how many people begin their shift at 12 PM. With this formulation, we see that the number of employees used is $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$. While this is the same formula as the incorrect model, the variables have very different values.

Now let's turn our attention to the constraints. For example, we need at least 8 employees to work between 12 AM and 4 AM. Who works during this time? Obviously, those who start at 12 AM work then, but further analysis shows that those who started at 8 PM are also working between 12 AM and 4 AM. Thus, to ensure that we have at least 8 employees working during this time, we have the constraint

$$x_1 + x_6 \geq 8.$$

Similar arguments for each of the remaining time periods give us Linear Program 2.2.

Linear Program 2.2 Work Schedule Model for Example 2.2

$$\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
\text{s.t.} \quad & \\
& x_1 + x_6 \geq 8 \quad (12 \text{ AM--4 AM}) \\
& x_1 + x_2 \geq 9 \quad (4 \text{ AM--8 AM}) \\
& x_2 + x_3 \geq 15 \quad (8 \text{ AM--12 PM}) \\
& x_3 + x_4 \geq 14 \quad (12 \text{ PM--4 PM}) \\
& x_4 + x_5 \geq 13 \quad (4 \text{ PM--8 PM}) \\
& x_5 + x_6 \geq 11 \quad (8 \text{ PM--12 AM}) \\
& x_i \geq 0, \quad i \in \{1, \dots, 6\}
\end{aligned}$$

Solution An optimal schedule utilizes only 36 employees and proportions them as follows:

Period	Number of Employees Beginning Shift
12 AM–4 AM	8
4 AM–8 AM	3
8 AM–12 PM	12
12 PM–4 PM	2
4 PM–8 PM	11
8 PM–12 AM	0

This solution indicates that every shift except for the 4 AM–8 AM shift has the minimum number of employees required. Thus, if we need to increase the required number of employees in any of these shifts, our total work force would need to increase as well.

In addition, note that this solution has integer-valued variables, even though we did not ask for it. In general, we cannot expect this to occur; however, it is often worthwhile computationally to see if we get integer solutions without stating the integrality constraints before we add these constraints. This is because, as we will see in Chapters 13 and 14, it is computationally more difficult (and time consuming) to solve problems with integer-constrained variables.

2.3 MODELS AND DATA

Now that we have seen a few linear programming models, let's take a step

back. Both of these models have very similar features. First, notice how the numbers given in each problem have little-to-no effect on the modeling process itself; if we alter any of the numbers in these examples, we do not affect the “look” of the optimization problem (we may change its optimal value, though—this is different!). Second, it was probably obvious to you that many of the constraints had the same structure, even though they were based on different restrictions of the problem. It is these two characterizations of the example models that are used to create more general representations of our problems.

Even though the examples in the first two sections of this chapter were small, it is not difficult to envision similar problems having many more variables and constraints. In Example 2.1, it is easy to envision a scenario where there are 30 door types instead of 3 and 25 machines instead of 2. However, we can probably surmise that the constraints corresponding to available machine times would look very similar. In Example 2.2, a similar notion occurs, since each constraint corresponds to a particular work schedule. What if there were other scheduling options, such as someone working only one 4-hour shift or working 12-hour shifts; could we easily incorporate such an option into our model?

What does this constraint similarity give us? Let’s consider Example 2.1. We initially constructed the constraints

$$3.5x_1 + 6x_2 + 8x_3 \leq 120$$

$$4x_1 + 5x_2 + 6x_3 \leq 100$$

corresponding to each machine’s hour requirements, where the variables corresponded to the amounts of each door type to produce. Both of these constraints have exactly the same structure: if we let $DoorTypes = \{\text{“Standard,” “High Security,” “Maximum Security”}\}$ be the set of door types and $MachineTypes = \{1, 2\}$ be the set of machinetypes, then, for each $j \in MachineType$, the constraints can be written in the form

$$(2.1) \sum_{j \in DoorTypes} t_{ij} x_j \leq maxhrs_i,$$

where t_{ij} corresponds to the number of hours door type $j \in DoorTypes$ needs on machine $i \in MachineType$ and $maxhrs_i$ indicates the maximum number of hours available on machine i . This algebraic formulation of our constraints

gives us many advantages. Since all the machine requirement constraints are of this form, we can easily increase the number of door types and machine types without altering the formulation of the problem; all we would need is to provide the values for each parameter t_{ij} and maxhrs_i . Second, by writing the constraints in this general form, we have separated out the actual model of the problem from its data parameters. Thus, the parameters can be altered while still maintaining the inherent mathematical structure of the model formulation. Finally, writing constraints in these general forms allows us to focus on the various classes of constraints that often occur many times within a given model; for example, constraints such as (2.1) representing the time availability of each machine can be combined with constraints of the form

$$x_j \leq \text{MaxAmt}_j$$

(which indicate the maximum amount of each door type we are allowed to produce) to provide a complete model of a problem using only two classes of constraints, even though each class may represent many individual constraints (one for each machine type or door type).

Because of the usefulness and efficiency of these algebraic constraint forms, *Modeling Languages* have been developed to translate these general modeling forms into specific instances solved by an optimization software package. There are currently many different modeling languages available to us; these include, for example, AMPL, GAMS, MPL, Mosel, and OPL. Some of these modeling languages, such as AMPL, GAMS, and MPL, can be used with a variety of optimization packages, while others such as Mosel and OPL are designed to work with specific solvers. They each also have their own way of representing the same constraints, although they often use very similar syntax. For example, constraints (2.1) could be written in AMPL as

```
subject to MachHrs { i in Machines } : sum {j in DoorTypes} t[i, j] * x[j]
<= maxhrs[i];
```

while the same constraints in Mosel would look like

```
forall(i in Machines)  MachHrs(i) := sum(j in DoorTypes) t(i, j) * x(j)
<= maxhrs(i)
```

In addition, modeling languages also include methods to access data, alter models when desired, call the appropriate solver package, and produce summary reports to help analyze the obtained solution.

In the forthcoming examples in the rest of this chapter and in Chapter 3, we highlight the general form of each model. This may be the entire model, so that we can see how to write general models mathematically, or just specific constraints that often appear.

2.4 BLENDING MODELS

One of the earliest linear programming models used in industry came from the petroleum industry. They were interested in creating gasoline and other oil products from crude oil in the most cost-efficient manner possible. Such *Blending Models* arise in many different areas:

1. Blending various chemicals to produce other chemicals.
2. Blending various alloys to produce steel.
3. Blending ingredients to produce certain foods.

Below is a classical example of this type of model.

■ EXAMPLE 2.3

Hoosier Gasoline Company, or HoosCo as they are known, produces two blends of gasoline, regular and premium, by mixing three different types of oil, indicated by Types 1, 2, and 3. Each type of oil comes in barrels and has its own costs and octane readings, which are given below.

	Cost/Barrel	Octane Rating	Maximum Available
Type 1	\$45	93	10,000
Type 2	\$35	91	15,000
Type 3	\$20	87	20,000

To produce each blend of gasoline, we must combine these oils so as to meet the following requirements.

	At least 25% Type 1
Regular gasoline	At least 35% Type 2
	At most 30% Type 3
	At least 30% Type 1
Premium gasoline	At least 45% Type 2
	At most 20% Type 3

There are also sales prices, average octane rating requirements for the gasolines, and minimum requirements in production of each blend.

	Average Octane Rating	Sales Price	Minimum Demand
Regular gasoline	89	\$65	15,000 barrels
Premium gasoline	91	\$75	12,500 barrels

How much of each type of gasoline should be produced to satisfy all requirements and maximize profits?

The important part of formulating this model is determining what the decision variables are. Here it is useful to define the variables as

x_{ij} = barrels of oil type i used to produce gasoline type j ,

q_i = barrels of oil type i used,

z_j = barrels of gasoline type j produced.

for $i \in \{1, 2, 3\}$ and $j \in \{1, 2\}$, where gas type 1 is regular and gas type 2 is premium. For example, x_{21} indicates the number of barrels of oil type 2 used to produce regular gasoline. We use double-indexed variables, which is a common technique when we want to keep track of the relationship between two classes of indices. In this case, we're interested in the relationship between oil types and gasoline types.

Having these definitions in place, we can determine that

$$q_1 = x_{11} + x_{12} = \text{barrels of oil type 1 used}$$

$$q_2 = x_{21} + x_{22} = \text{barrels of oil type 2 used}$$

$$q_3 = x_{31} + x_{32} = \text{barrels of oil type 3 used}$$

$$z_1 = x_{11} + x_{21} + x_{31} = \text{barrels of regular gasoline produced}$$

$$z_2 = x_{12} + x_{22} + x_{32} = \text{barrels of premium gasoline produced.}$$

Knowing this, our profit can be calculated, using the fact that profit is revenue minus cost:

$$\begin{aligned} \text{profit} &= 75(x_{11} + x_{21} + x_{31}) + 65(x_{12} + x_{22} + x_{32}) \\ &\quad - [45(x_{11} + x_{12}) + 35(x_{21} + x_{22}) + 20(x_{31} + x_{32})] \\ &= 75z_1 + 65z_2 - (45q_1 + 35q_2 + 20q_3). \end{aligned}$$

We also know, from available oil and minimum demand, that the following inequalities must hold in our solution:

$$q_1 \leq 10,000$$

$$q_2 \leq 15,000$$

$$q_3 \leq 20,000$$

$$z_1 \geq 15,000$$

$$z_2 \geq 12,500.$$

Now we consider the other constraints. How can we specify that at least 25% of regular gasoline is made of type 1 oil? This almost seems to indicate a ratio of some sort, which is not linear. However, we can use this knowledge to get a linear constraint. For example, the percentage of type 1 oil in regular gasoline is equal to the number of barrels of type 1 oil divided by the total number of barrels of regular gasoline produced, that is,

$$\frac{x_{11}}{x_{11} + x_{21} + x_{31}} = \frac{x_{11}}{z_1} = \frac{\text{number of barrels of type 1 oil in regular gasoline}}{\text{number of barrels of regular gasoline produced}}.$$

Hence, our constraint is

$$\frac{x_{11}}{x_{11} + x_{21} + x_{31}} \geq 0.25$$

or

$$x_{11} \geq 0.25(x_{11} + x_{21} + x_{31})$$

or

$$x_{11} \geq 0.25z_1.$$

Similarly, to specify that at most 30% of regular gasoline is made of type 3 oil, we'd have that

$$\frac{x_{31}}{x_{11} + x_{21} + x_{31}} \leq 0.3$$

or

$$x_{31} \leq 0.3(x_{11} + x_{21} + x_{31})$$

or

$$x_{31} \leq 0.3z_1.$$

We can generate the other constraints of this class using this same technique, giving

$$x_{21} \geq 0.35z_1$$

$$x_{12} \geq 0.3z_2$$

$$x_{22} \geq 0.45z_2$$

$$x_{32} \leq 0.2z_2.$$

Note that we can multiply through by the denominator in each ratio because we are assuming that a positive amount of each gasoline will be produced.

To find the average octane rating of each gasoline, we use the fact that if, say, 40% of regular gasoline comes from oil type 1, then 40% of its average octane rating comes from the octane rating of oil type 1. Using similar arguments to those above, we have the following constraints

$$(93) \frac{x_{11}}{x_{11} + x_{21} + x_{31}} + (91) \frac{x_{21}}{x_{11} + x_{21} + x_{31}} + (87) \frac{x_{31}}{x_{11} + x_{21} + x_{31}} \geq 89$$

$$(93) \frac{x_{12}}{x_{12} + x_{22} + x_{32}} + (91) \frac{x_{22}}{x_{12} + x_{22} + x_{32}} + (87) \frac{x_{32}}{x_{12} + x_{22} + x_{32}} \geq 91$$

for regular gasoline and premium gasoline, respectively. These can be rewritten as follows:

$$93x_{11} + 91x_{21} + 87x_{31} \geq 89z_1 \quad (\text{regular gasoline}),$$

$$93x_{12} + 91x_{22} + 87x_{32} \geq 91z_2 \quad (\text{premium gasoline}).$$

Putting this all together, we have Linear Program 2.3 (written without using variables q_i and z_j so that you can see the full effect of the blending constraints).

Linear Program 2.3 Blending Model for Example 2.3

$$\begin{aligned}
\max \quad & 75(x_{11} + x_{21} + x_{31}) + 65(x_{12} + x_{22} + x_{32}) \\
& - 45(x_{11} + x_{12}) - 35(x_{21} + x_{22}) - 20(x_{31} + x_{32}) \\
\text{s.t.} \quad & x_{11} \geq 0.25(x_{11} + x_{21} + x_{31}) \\
& x_{21} \geq 0.35(x_{11} + x_{21} + x_{31}) \\
& x_{31} \leq 0.3(x_{11} + x_{21} + x_{31}) \\
& x_{12} \geq 0.3(x_{12} + x_{22} + x_{32}) \\
& x_{22} \geq 0.45(x_{12} + x_{22} + x_{32}) \\
& x_{32} \leq 0.2(x_{12} + x_{22} + x_{32}) \\
& 93x_{11} + 91x_{21} + 87x_{31} \geq 89(x_{11} + x_{21} + x_{31}) \\
& 93x_{12} + 91x_{22} + 87x_{32} \geq 91(x_{12} + x_{22} + x_{32}) \\
& x_{11} + x_{12} \leq 10,000 \\
& x_{21} + x_{22} \leq 15,000 \\
& x_{31} + x_{32} \leq 20,000 \\
& x_{11} + x_{21} + x_{31} \geq 15,000 \\
& x_{12} + x_{22} + x_{32} \geq 12,500 \\
& x_{ij} \geq 0, \quad i \in \{1, 2, 3\}, j \in \{1, 2\}
\end{aligned}$$

Solution An optimal solution would be to purchase 10, 000 barrels of crude oil 1, 8750 barrels of crude oil 2, and 16, 250 barrels of crude oil 3. These barrels are blended as

	Crude Oil 1	Crude Oil 2	Crude Oil 3	Total
Regular	5625	2812.5	14,062.5	22,500
Premium	4375	5937.5	2187.5	12,500

where the optimal profit would be \$1,318,750. Here, a fractional solution seems reasonable, since we can consider taking fractions of barrels for each type of gas. The only variables we may consider making integer-valued would be the total number of barrels of each crude oil purchased and the total number of barrels of each gasoline sold; since this solution already satisfies these restrictions, we do not add them to the model. Note that, if we were to add these constraints, we would have to explicitly include variables q_i and z_j into our model.

One interesting element of this solution is that, in order to satisfy all the blending and demand requirements, we actually produce more regular

gasoline than is needed. If we assume that we can sell the extra gasoline at the same price of \$75 per barrel, this causes no problem. However, if we do not have that much extra demand for regular gasoline, then we must have our model consider this. For example, we could simply change the demand constraints into equality constraints

$$\begin{aligned}x_{11} + x_{21} + x_{31} &= 15,000 \\x_{12} + x_{22} + x_{32} &= 12,500.\end{aligned}$$

This results in an optimal profit of \$1,042,190, which is considerably less than that before. By forcing the demand constraints to be equalities, we are severely limiting the feasible solutions. Better alternatives would be to put both a lower and an upper bound on the amount of gasoline to produce, or to penalize “overproduction” of gasoline, while still allowing it (see Exercise 2.15).

General Blending Model Constraints Blending models present themselves nicely into generalized forms. In fact, we have already discussed some of these forms in the specific instance above. However, for completeness, it makes sense to state them for posterity.

Since blending models need not involve oil and gasoline, let's suppose that we are combining some ingredients I in order to produce products J . Each product $j \in J$ is created by simply combining various ingredients $i \in I$; the total amount of product j is computed by adding the total amounts of the ingredients. It is therefore natural to define, for each $i \in I$ and $j \in J$,

$$x_{ij} = \text{amount of ingredient } i \text{ in product } j.$$

As we did previously, we can easily compute the total amount of each ingredient i used and the total amount of each product j produced. By defining

$$q_i = \text{total amount of ingredient } i \text{ used},$$

$$z_j = \text{total amount of product } j \text{ produced},$$

we have

$$q_i = \sum_{j \in J} x_{ij}, \quad i \in I$$

$$z_j = \sum_{i \in I} x_{ij}, \quad j \in J.$$

Since there are typically a maximum amount \maxIng_i of each ingredient i available for use and a minimum required amount \minDemand_j for each product j ,

we have the constraints

$$\begin{aligned} q_i &\leq \text{maxIng}_i, & i \in I \\ z_j &\geq \text{demand}_j, & j \in J. \end{aligned}$$

The blending constraints are also similar to those constructed earlier.

Suppose that product j must include at least some proportion p_{ij} of ingredient i . Since the nonlinear relationship would naturally be

$$\frac{x_{ij}}{\sum_{k \in I} x_{kj}} \geq p_{ij}$$

we would rewrite this as the linear constraint

$$x_{ij} \geq p_{ij} \sum_{k \in I} x_{kj}$$

or more succinctly as

$$x_{ij} \geq p_{ij} z_j.$$

Similar thinking would generate the constraints indicating a maximum percentage p_{ij} of ingredient i in product j :

$$x_{ij} \leq p_{ij} z_j.$$

Such **Blending Constraints** occur frequently and in diverse applications, including (but not limited to) oil and gas production, food production, and chemical mixes for farming.

2.5 PRODUCTION PROCESS MODELS

Often, especially in production, a process goes through many different stages, where the output of a later stage depends on the output from an earlier stage. Linear programs are often used successfully to handle such instances, as illustrated in the following example.

■ EXAMPLE 2.4

The ChocoChew Company has been selling chewy chocolate candies for over

a century. Their candies come in various sizes—long and thin, medium and thick, midget, and even inside lollipops (ChocoPops). Each type can be sold separately or collectively in large packages, and all types (except for the ChocoPops) are the same candies but in different sizes. Initially, during production, both thin and thick ChocoChews are produced. Because each candy type is identical except for its shape and size, they are all rolled using the same raw materials, which are the ingredients already mixed together. A pound of raw material can produce either 100 thick ChocoChews, which takes 0.002 minutes, or 250 thin ChocoChews, which takes 0.004 minutes. The ingredients combined cost \$0.2 per pound, and it costs the company \$0.01 per pound of raw material to produce 100 thick ChocoChews and \$0.015 per pound to produce 250 thin ChocoChews. The midget-sized ChocoChews are then produced by cutting either a thin ChocoChew into thirds (which takes 0.00015 minutes and costs \$0.0001 per thin ChocoChew in reprocessing) or a thick ChocoChew into quarters (which takes 0.00015 minutes and costs \$0.00005 in reprocessing), while the ChocoPops are produced by first cutting a thick ChocoChew into five equal-sized pieces and then enveloping each inside a candy shell (which takes 0.0003 minutes and costs \$0.004 in ingredients and processing). There are daily demands for each candy sold in separate packages and also sold in the collective multipacks; this information is given in the table below.

Product	Daily Demands	
	Individual Package	Multipack
Thin ChocoChews	2,000,000	2,000,000
Thick ChocoChews	1,500,000	1,000,000
Midget ChocoChews	5,000,000	3,000,000
ChocoPops	1,000,000	1,500,000

Assuming that there are only 12 working hours in a day, how can the ChocoChew Company determine an optimal production schedule in order to meet its daily demands at minimum cost?

In such production process models, we need to keep track of quantities from the various stages of production. Here, we have raw material (stage 1) being converted into Thin ChocoChews or Thick ChocoChews (stage 2), which can be further processed into either Midget ChocoChews or ChocoPops (stage 3); in addition, each candy type can be sold individually or as part of a

multipack. Because of this, we need to break down our variables into those for each stage and those for the transition from one stage to the next. For example, for stage 1 we define the variable *Raw* to equal the total number of pounds of raw material used in the production process. Variable *Raw* can be further refined into the amount of raw material used for either Thin ChocoChews (*RawToThin*) or Thick ChocoChews (*RawToThick*), which are the transition variables between stages 1 and 2; these variables are related by

$$\underline{2.2} \quad \text{Raw} = \text{RawToThin} + \text{RawToThick}.$$

Now that our model has partitioned the raw material into those used for the Thin and Thick ChocoChews, we proceed to stage 2, which processes it into the actual candies. We can handle this by constructing the variables *ThinProduced* and *ThickProduced* to indicate how much of each candy is produced from raw material. These are calculated by

$$(2.3) \quad \text{ThinProduced} = 250\text{RawToThin}$$

$$(2.4) \quad \text{ThickProduced} = 100\text{RawToThick}.$$

Obviously, we can combine (2.3) and (2.4) with [equation \(2.2\)](#) to generate one equation, but the logic of the model would not be as evident then. Fortunately, most commercial optimization solvers will do this prior to actually solving the problem (in order to reduce the number of variables and/or constraints), so we often do not need to worry about this when modeling.

Now that we have the stage 2 quantities, we need to do further refining. Each Thin and Thick ChocoChew produced can be either sold as part of individual packages or multipacks, or further processed into Midget ChocoChews or, in the case of the Thick ChocoChews, ChocoPops. To handle this situation, we can add another set of variables *SoldThin* and *SoldThick* to indicate those candies that are to be sold (in one form or another) and variables *ThinToMidget*, *ThickToMidget*, and *ThickToPops* that give us the number of Thin and Thick ChocoChews that are further processed. These variables are related by the equations

$$\text{ThinProduced} = \text{SoldThin} + \text{ThinToMidget}$$

$$\text{ThickProduced} = \text{SoldThick} + \text{ThickToMidget} + \text{ThickToPops}.$$

Having now determined how many Thin and Thick ChocoChews are used to produce Midget ChocoChews and ChocoPops, we can calculate the actual

number of these candies produced, using the variables *MidgetsProduced* and *PopsProduced* along with the equations

$$MidgetsProduced = 3ThinToMidget + 4ThickToMidget$$

$$PopsProduced = 5ThickToPops.$$

Finally, we can determine how many of each candy are sold in the various packages; note that, since Midget ChocoChews and ChocoPops are not further processed into other candies, the variables *MidgetsProduced* and *PopsProduced* also indicate the total number of each sold. To do so, we will need two additional variables for each candy type (*SoldInd* and *SoldMulti*), giving the amounts sold in each type of package. This then leads to the equations

$$SoldThin = SoldIndThin + SoldMultiThin$$

$$SoldThick = SoldIndThick + SoldMultiThick$$

$$MidgetsProduced = SoldIndMidgets + SoldMultiMidgets$$

$$PopsProduced = SoldIndPops + SoldMultiPops.$$

Now that we have determined how each candy type is processed, we need to constrain the total amount of hours needed in this process; since processing times are given in minutes, our constraint should also be written in minutes:

$$\begin{aligned} & 0.002RawToThin + 0.004RawToThick + 0.00015ThinToMidget \\ & + 0.00015ThickToMidget + 0.0003ThickToPops \leq 12 * 60 = 720. \end{aligned}$$

Our last set of constraints provides the minimum amounts for each type that needs to be produced daily:

$$SoldIndThin \geq 2,000,000$$

$$SoldMultiThin \geq 2,000,000$$

$$SoldIndThick \geq 1,500,000$$

$$SoldMultiThick \geq 1,000,000$$

$$SoldIndMidgets \geq 5,000,000$$

$$SoldMultiMidgets \geq 3,000,000$$

$$SoldIndPops \geq 1,000,000$$

$$SoldMultiPops \geq 1,500,000.$$

Now it's time to write our objective function. Our costs can be broken down into raw material costs and production costs. The cost associated with raw materials is $0.2Raw$, since each pound of raw material costs \$0.2. Production

costs depend on which candy is being produced and from what source. For example, the cost to produce Midget ChocoChews from Thin ChocoChews will total $0.0001ThinToMidget$, while the cost to produce Midget ChocoChews from Thick ChocoChews will total $0.00005 ThickToMidget$. The total cost associated with this model is

$$\begin{aligned}Cost = & 0.2Raw + 0.01RawToThick + 0.015RawToThin \\& + 0.0001ThinToMidget + 0.00005ThickToMidget \\& + 0.004ThickToPops.\end{aligned}$$

The complete optimization model for the ChocoChew Company is given in Linear Program 2.4.

Linear Program 2.4 Production Process Model for Example 2.4

$$\begin{aligned}
\min \quad & 0.2Raw + 0.01RawToThick + 0.015RawToThin \\
& + 0.0001ThinToMidget + 0.00005ThickToMidget \\
& + 0.004ThickToPops \\
\text{s.t.} \quad & Raw = RawToThin + RawToThick \\
& ThinProduced = 250RawToThin \\
& ThickProduced = 100RawToThick \\
& ThinProduced = SoldThin + ThinToMidget \\
& ThickProduced = SoldThick + ThickToMidget + ThickToPops \\
& MidgetsProduced = 3ThinToMidget + 4ThickToMidget \\
& PopsProduced = 5ThickToPops \\
& SoldThin = SoldIndThin + SoldMultiThin \\
& SoldThick = SoldIndThick + SoldMultiThick \\
& MidgetsProduced = SoldIndMidgets + SoldMultiMidgets \\
& PopsProduced = SoldIndPops + SoldMultiPops \\
& 0.002RawToThin + 0.004RawToThick + 0.00015ThinToMidget \\
& + 0.00015ThickToMidget + 0.0003ThickToPops \leq 720 \\
& SoldIndThin \geq 2,000,000 \\
& SoldMultiThin \geq 2,000,000 \\
& SoldIndThick \geq 1,500,000 \\
& SoldMultiThick \geq 1,000,000 \\
& SoldIndMidgets \geq 5,000,000 \\
& SoldMultiMidgets \geq 3,000,000 \\
& SoldIndPops \geq 1,000,000 \\
& SoldMultiPops \geq 1,500,000
\end{aligned}$$

Did you notice that, in this model, we assumed that it was acceptable to produce more of any type of candy than their minimum requirement. Also, we made no assumptions regarding the integrality of the number of any candy type. This may need to be addressed in our model if the optimal solution contains fractional values. However, it does seem reasonable to have a fractional amount of raw material.

Solution An optimal solution to this model would be to use 57,419.3548 pounds of raw material to produce 6,451,612.9032 Thin ChocoChews and 3,161,290.3226 Thick ChocoChews. From 2,451,612.9032 Thin

ChocoChews and 161,290.3226 Thick ChocoChews we would produce 8,000,000 Midget ChocoChews, and from 500,000 Thick ChocoChews we would produce 2,500,000 ChocoPops. We would sell the minimum number of candies required by the current demand. Our total production cost would be \$14,440.3226.

If we require that the number of candies produced and sold take integer values, our solution changes slightly. We would use 57,419.3580 pounds of raw material, from which we would produce 6,451,612 Thin ChocoChews and 3,161,291 Thick ChocoChews; note that these quantities are not the “rounded” values from the previous solution. The total cost changes by less than a penny to \$14,440.3232.

Another interesting aspect to this problem is that, if we were able to extend the production time by 5 minutes, we could reduce our total cost by over \$100. This is the optimal solution if time is not a constraint. In addition, we find that we can satisfy all demand using only 682 minutes, or 11 hours and 22 minutes, but our costs increase by about \$1600. This tells us that our demands are close to the current capacity we can handle under the current situation. Any change in the demand requires change in our production capabilities.

General Model Now let’s construct a general model for this problem. We can easily see that there are a number of constraints relating the production of one type of candy to another type. To write such constraints more generally, we will define several subsets of products and new variables to indicate these productions. First, let’s partition the products (P) into (a) those that are produced directly from raw material (FR), (b) those derived only from other products and can be used to further produce additional products (MP), and (c) those that are only sold (FP); in our specific example, we have $P = \{Thin, Thick, Midgets, Pops\}$, $FR = \{Thin, Thick\}$, $MP = \emptyset$, and $FP = \{Midgets, Pops\}$. For each product $k \in P$, we define the variables

$$Amt_k = \text{amount of product } k \text{ produced},$$

$$Sold_k = \text{amount of product } k \text{ sold}.$$

We also define a variable Amt_{Raw} to indicate the amount of raw material used. For the item $i \in \{raw\} \cup FR \cup MP$ (i.e., those items from which other products are produced) and each product $j \in MP \cup FP$, let’s define

$Produce_{ij}$ = amount of item i used to produce product j ,

$rate_{ij}$ = amount of product j produced per item i (parameter).

With these variables and the parameters $rate_{ij}$, we can define our constraints by

$$\begin{aligned} Amt_k &= rate_{raw,k} Produce_{raw,k}, & k \in FR \\ Amt_k &= Sold_k + \sum_{i \in MP} Produce_{ki} + \sum_{j \in FP} Produce_{kj}, & k \in FR \\ Amt_j &= \sum_{i \in MP} rate_{ij} Produce_{ij}, & j \in MP \\ Amt_j &= Sold_j + \sum_{i \in MP} Produce_{ji} + \sum_{k \in FP} Produce_{jk}, & j \in MP \\ Amt_j &= \sum_{i \in MP} rate_{ij} Produce_{ij}, & j \in FP \\ Amt_j &= Sold_j, & j \in FP. \end{aligned}$$

Now, we need to add additional variables $SoldInd_k$ and $SoldMulti_k$ for each product $k \in P$ to indicate how each piece is to be sold:

$$Sold_k = SoldInd_k + SoldMulti_k.$$

Our last constraints indicate the minimum requirements for each product $k \in P$:

$$\begin{aligned} SoldInd_k &\geq MinInd_k \\ SoldMulti_k &\geq MinMulti_k, \end{aligned}$$

where $MinInd_k$ and $MinMulti_k$ are parameters indicating the appropriate minimum requirements for product k .

Finally, we need to consider the total cost. If we let

$$c_{ij} = \text{cost of processing item } i \text{ into item } j,$$

then our total cost can be calculated as

$$\begin{aligned} Cost &= \sum_{j \in FR} c_{raw,j} Produce_{raw,j} + \sum_{i \in FR} \sum_{j \in MP} c_{ij} Produce_{ij} \\ &\quad + \sum_{i \in FR} \sum_{j \in FP} c_{ij} Produce_{ij} + \sum_{i \in MP} \sum_{j \in MP} c_{ij} Produce_{ij} \\ &\quad + \sum_{i \in MP} \sum_{j \in FP} c_{ij} Produce_{ij}. \end{aligned}$$

Putting this all together gives a final model given in Linear Program 2.5.

Linear Program 2.5 General Model for ChocoChews Problem

$$\begin{aligned}
\min \quad & \sum_{j \in FR} c_{raw,j} Produce_{raw,j} + \sum_{i \in FR} \sum_{j \in MP} c_{ij} Produce_{ij} \\
& + \sum_{i \in FR} \sum_{j \in FP} c_{ij} Produce_{ij} + \sum_{i \in MP} \sum_{j \in MP} c_{ij} Produce_{ij} \\
& + \sum_{i \in MP} \sum_{j \in FP} c_{ij} Produce_{ij} \\
\text{s.t.} \quad & Amt_k = rate_{raw,k} Produce_{raw,k}, \quad k \in FR \\
& Amt_k = Sold_k + \sum_{i \in MP} Produce_{ki} + \sum_{j \in FP} Produce_{kj}, \quad k \in FR \\
& Amt_j = \sum_{i \in MP} rate_{ij} Produce_{ij}, \quad j \in MP \\
& Amt_j = Sold_j + \sum_{i \in MP} Produce_{ji} + \sum_{k \in FP} Produce_{jk}, \quad j \in MP \\
& Amt_j = \sum_{i \in MP} rate_{ij} Produce_{ij}, \quad j \in FP \\
& Amt_j = Sold_j, \quad j \in FP \\
& Sold_k = SoldInd_k + SoldMulti_k, \\
& SoldInd_k \geq MinInd_k, \quad k \in P \\
& SoldMulti_k \geq MinMulti_k,
\end{aligned}$$

2.6 MULTIPERIOD MODELS: WORK SCHEDULING AND INVENTORY

As we saw in Section 2.5, linear programs can be used to model multistage problems. We now expand this multistage approach into other *multiperiod* models. These models arise when decisions are made for more than one period, such as determining production processes and inventory levels over multiple months.

■ EXAMPLE 2.5

American Engine Co. produces two engines, one for trucks and one for cars. During the next 3 months, they anticipate the following demands for their engines:

	Month 1	Month 2	Month 3
Truck engines	400	300	500
Car engines	800	500	600

No backlogging is allowed, which means that each month's demand must be fully satisfied. During each month, at most 1000 engines (combined) can be produced. Each truck engine requires 10 hours of labor to produce and costs \$2000 in supplies, while each car engine requires 8 hours of labor and costs \$1500 in supplies. At most 9000 hours are available each month. At the beginning of month 1, 100 truck engines and 200 car engines are in inventory. At the end of each month, a holding cost of \$150 per engine is assigned to any engine in inventory. At the end of the third month, management wants to have at least 100 of each engine in inventory. How can we meet monthly demand at a minimum cost?

We want to identify, for each month, how many of each engine is produced and how many engines are placed in inventory. To this end, we define the following decision variables for each month $i \in \{1, 2, 3\}$:

T_i = number of truck engines produced during month i ,

C_i = number of car engines produced during month i ,

IT_i = number of truck engines in inventory at end of month i ,

IC_i = number of car engines in inventory at end of month i .

We will also define, as parameters, the values $IT_0 = 100$ and $IC_0 = 200$, that is, the starting inventory levels for each engine type. Similar to earlier models, the constraints on the number of engines produced and the labor used in month $i \in \{1, 2, 3\}$ are

$$T_i + C_i \leq 1000$$

$$10T_i + 8C_i \leq 9000.$$

The inventory constraints are new to us, but are easier to identify once we've made the following two observations:

- Inventory at end of month t = inventory at end of month $(t - 1)$
+ production during month t
– demand during month t .
- Inventory at end of month $t \geq 0$, if no backlog is allowed.

Having made these observations, we have the following inventory constraints:

$$IT_1 = 100 + T_1 - 400$$

$$IT_2 = IT_1 + T_2 - 300$$

$$IT_3 = IT_2 + T_3 - 500$$

$$IC_1 = 200 + C_1 - 800$$

$$IC_2 = IC_1 + C_2 - 500$$

$$IC_3 = IC_2 + C_3 - 600.$$

We have assumed that there is enough space in inventory to hold as many engines as needed. Recall that we also wanted to have our final inventory levels IC_3 and IT_3 to each be at least 100.

Our objective function seeks to minimize the total cost. Our costs in this problem are (1) holding costs associated with inventory levels and (2) costs for supplies. To calculate holding costs, since each engine in inventory costs the same amount of money, namely \$150, we have

$$\text{holding costs} = 150 (IT_1 + IT_2 + IT_3 + IC_1 + IC_2 + IC_3).$$

Our supply costs are

$$\text{supply cost} = 2000 (T_1 + T_2 + T_3) + 1500(C_1 + C_2 + C_3).$$

Our final linear program is given in Linear Program 2.6. You may wonder why we included the variable bounds $IC_3, IT_3 \geq 0$ when they are redundant. We include them because we will assume that all variables have nonnegativity bounds (≥ 0) when we consider solving linear programs, even if there are other lower bounds. This is not really a problem, unless of course we have variables that have bounds like $x \geq -2$.

Linear Program 2.6 Inventory Model for Example 2.5

$$\begin{aligned}
\min \quad & 150 (IT_1 + IT_2 + IT_3 + IC_1 + IC_2 + IC_3) \\
& + 2000 (T_1 + T_2 + T_3) + 1500 (C_1 + C_2 + C_3) \\
\text{s.t.} \quad & T_i + C_i \leq 1000, \quad i \in \{1, 2, 3\} \\
& 10T_i + 8C_i \leq 9000, \quad i \in \{1, 2, 3\} \\
& IT_1 = 100 + T_1 - 400 \\
& IT_2 = IT_1 + T_2 - 300 \\
& IT_3 = IT_2 + T_3 - 500 \\
& IC_1 = 200 + C_1 - 800 \\
& IC_2 = IC_1 + C_2 - 500 \\
& IC_3 = IC_2 + C_3 - 600 \\
& IC_3 \geq 100, IT_3 \geq 100 \\
& T_1, T_2, T_3, C_1, C_2, C_3, \\
& IT_1, IT_2, IT_3, IC_1, IC_2, IC_3 \geq 0
\end{aligned}$$

Solution An optimal solution has total cost of \$5,190,000 with values given in the table below.

Month	Engine Type	Production Amt	Inventory Level
Month 1			
	Car	600	0
	Truck	400	100
Month 2			
	Car	500	0
	Truck	500	300
Month 3			
	Car	700	100
	Truck	300	100

During each month we produce as many engines as possible, and only during month 2 do we use all the available labor hours. It would seem reasonable to explore the effect of increasing the total production amount limit from 1000 to see if we can reduce our cost further; it probably is not worth exploring adding more labor hours.

General Model As this example illustrates, multiperiod constraints typically have a general mathematical form that is easy to state.

Inventory-type constraints often have the form

$$(2.5) \quad I_t = I_{t-1} + C_t - D_t,$$

for each time period t , where I_t is the inventory level for period t , C_t is the amount of product generated during period t , and D_t is the demand for period t . Care must be taken for the initial time period, however; we typically define the initial inventory level I_0 , even though there is no “real” period 0.

In fact, these inventory-type constraints can be generalized further. Note that the quantity $C_t - D_t$ in (2.5), which is the production amount minus the demand, reflects the change in inventory level from one period to the next. Thus, we can state (2.5) more generally as

$$(2.6) \quad \left(\begin{array}{c} \text{Amount in} \\ \text{period } t \end{array} \right) = \left(\begin{array}{c} \text{Amount in} \\ \text{period } t-1 \end{array} \right) + \left(\begin{array}{c} \text{Change during} \\ \text{period } t \end{array} \right).$$

In addition, this change can be positive, negative, or zero. If we had simply rewritten (2.6) as

$$\left(\begin{array}{c} \text{Change during} \\ \text{period } t \end{array} \right) = \left(\begin{array}{c} \text{Amount in} \\ \text{period } t \end{array} \right) - \left(\begin{array}{c} \text{Amount in} \\ \text{period } t-1 \end{array} \right),$$

we see that we denote a quantity (Change) that is not restricted in sign (can be positive, negative, or zero) to equal the difference of two nonnegative quantities (amounts in successive periods). This is a useful modeling construct, which we explore further in the next section.

2.7 LINEARIZATION OF SPECIAL NONLINEAR MODELS

Sometimes situations arise where linear models do not seem appropriate, such as when modeling the distance between two variable locations. Fortunately, there are some important nonlinear models that can be changed into linear models using some “tricks.” These models include the use of absolute values and minimax or maximin objective functions. This next example explores a few of these.

■ EXAMPLE 2.6

An engineering project needs to place three sensors in the ceiling, each connected to a central reporting system. Due to the design of the ceiling, the wiring between the sensors can run only along horizontal or vertical tracks in the ceiling. A map looking like a coordinate system describes where the sensors are to be placed. If the “origin” is in the southwest corner of the room, then the three sensors are located at $(10, 3)$, $(5, 15)$, and $(20, 25)$. Our job is to place the central unit. We are asked to come up with models that (1) minimize the total amount of wire used and (separately) (2) minimize the maximum wire length from a sensor to the central unit.

Since we want to find an ideal location of the central unit, it makes sense that our decision variables be

$$x = x\text{-coordinate of location}$$

$$y = y\text{-coordinate of location}.$$

Let’s first deal with the length of each wire. Typically, we would use the standard Euclidean distance formula (or l_2 -norm)

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

to compute the distance between points (x_0, y_0) and (x_1, y_1) . However, this assumes that the distance is measured in a straight line. In our case, we must place the wire so that it is parallel to both the horizontal and vertical “axes” (i.e., the walls).

To measure this distance, we note that, if the sensor is located at point (x_0, y_0) and our central unit is located at (x, y) , then the horizontal length of the wire is just $|x - x_0|$, and the vertical length is $|y - y_0|$. Hence, our wire length is $|x - x_0| + |y - y_0|$. Such a distance measure is known as either a *rectilinear distance* or a *Manhattan distance*; it is also referred to as the l_1 -norm. Our total distance is then

$$\text{wire length} = |x - 10| + |y - 3| + |x - 5| + |y - 15| + |x - 20| + |y - 25|.$$

If we want to minimize only our wire length, we could set up the following (nonlinear) mathematical program:

$$\begin{aligned} \min \quad & |x - 10| + |y - 3| + |x - 5| + |y - 15| + |x - 20| + |y - 25| \\ \text{s.t.} \quad & x, y \geq 0. \end{aligned} \tag{2.7}$$

We can also use a nonlinear program to find the location that minimizes the maximum wire length. In these situations, we're not so concerned with total wire length, but we want to have the unit centrally located. Thus, when minimizing the maximum wire length, we measure by the longest length rather than total length. In our case, fixing (for a moment) the unit location at (x, y) , the longest wire length is

$$\max \{|x - 10| + |y - 3|, |x - 5| + |y - 15|, |x - 20| + |y - 25|\}.$$

Our mathematical program is thus

$$\min \max \{|x - 10| + |y - 3|, |x - 5| + |y - 15|, |x - 20| + |y - 25|\}$$

s.t.

$$(2.8) \quad x, y \geq 0.$$

The objective of this mathematical program is a member of a larger class, which is defined below.

minimax and maximin Objective functions where we minimize the maximum of multiple functions are known as *minimax objective functions*, and objective functions where we maximize the minimum of multiple functions are known as *maximin objective functions*.

While many general mathematical programming algorithms can solve the two problems (2.7) and (2.8), they can also be transformed into linear programs. Let's first deal with the absolute value function. How can we transform $|w|$ into a linear function? We know

$$|w| = \begin{cases} w, & \text{if } w \geq 0, \\ -w, & \text{if } w < 0. \end{cases}$$

that One way to model this linearly is to borrow an idea first suggested in Section 2.6. There, we noted that the difference between production amount C_t and demand D_t during period t is unrestricted in sign and is equal to the difference in (nonnegative) inventory amounts between two periods. To extend this, suppose we have an unrestricted quantity (variable or function) w and we define nonnegative variables w^+ and w^- and add the constraint

$$w = w^+ - w^-.$$

Our hope is that at least one of the variables w^+ and w^- is zero, so that the

variables w^+ and w^- indicate a positive amount ($w^+ > 0, w^- = 0$), a negative amount ($w^+ = 0, w^- > 0$), or zero ($w^+ = w^- = 0$). We will see in Chapter 7 that, in most situations, an optimal solution to our linear program can be found where at least one of the variables w^+ and w^- is zero, provided an optimal solution exists.

Having dealt with modeling an unrestricted quantity w using nonnegative variables w^+ and w^- , can we write $|w|$ as a linear function of w^+ and w^- ? It should not be hard to see that

$$(2.9) |w| = w^+ + w^-$$

if at least one of the variables has a value of zero. For our specific problem, we'd have the following constraints:

$$\begin{aligned} x - 10 &= x_1^+ - x_1^- \\ y - 3 &= y_1^+ - y_1^- \\ x - 5 &= x_2^+ - x_2^- \\ y - 15 &= y_2^+ - y_2^- \\ x - 20 &= x_3^+ - x_3^- \\ y - 25 &= y_3^+ - y_3^-, \end{aligned}$$

where the subscripts indicate the sensor number. In the objective function we have the forms $w^+ + w^-$ for each quantity w within the absolute value. Given these constraints, and the fact that all variables are nonnegative, we'd have the linearized version of (2.7) shown in Linear Program 2.7.

Linear Program 2.7 Linearized Version of Minimum Wire Problem (2.7)

$$\begin{aligned}
\min \quad & (x_1^+ + x_1^-) + (y_1^+ + y_1^-) + (x_2^+ + x_2^-) + (y_2^+ + y_2^-) \\
& + (x_3^+ + x_3^-) + (y_3^+ + y_3^-) \\
\text{s.t.} \quad & x - 10 = x_1^+ - x_1^- \\
& y - 3 = y_1^+ - y_1^- \\
& x - 5 = x_2^+ - x_2^- \\
& y - 15 = y_2^+ - y_2^- \\
& x - 20 = x_3^+ - x_3^- \\
& y - 25 = y_3^+ - y_3^- \\
& x, y, x_1^+, x_1^-, y_1^+, y_1^-, x_2^+, x_2^-, y_2^+, y_2^-, x_3^+, x_3^-, y_3^+, y_3^- \geq 0
\end{aligned}$$

Solution The optimal position for the central unit is at (10, 15), which results in a total wire length of 37 units.

Now let's tackle the minimax problem. You should notice that all the constraints in Linear Program 2.7 will be used again in the minimax problem, since we still have the absolute value functions. The concern for minimax problems is in transforming the objective function into a linear function. One way we can do this is by introducing a new variable f and adding the following constraints:

$$\begin{aligned}
f &\geq (x_1^+ + x_1^-) + (y_1^+ + y_1^-) \\
f &\geq (x_2^+ + x_2^-) + (y_2^+ + y_2^-) \\
f &\geq (x_3^+ + x_3^-) + (y_3^+ + y_3^-).
\end{aligned}$$

Note that, if we minimize f , then f will equal the largest of these three values, which is what we want. Linear Program 2.8 gives the final model. Note that the maximin problem can be handled similarly.

Solution The optimal position of the central unit that minimizes the maximum wire length is (17.5, 11.5), which results in a maximum wire length of 16 units.

General Model We noted earlier in this section how to model quantities that are unrestricted in sign by equating them to the difference of two nonnegative variables. It is useful to state this again.

Linear Program 2.8 Linear Model for Minimax Wire Problem
(2.8)

$$\begin{aligned}
\min \quad & f \\
\text{s.t.} \quad & x - 10 = x_1^+ - x_1^- \\
& y - 3 = y_1^+ - y_1^- \\
& x - 5 = x_2^+ - x_2^- \\
& y - 15 = y_2^+ - y_2^- \\
& x - 20 = x_3^+ - x_3^- \\
& y - 25 = y_3^+ - y_3^- \\
& f \geq (x_1^+ + x_1^-) + (y_1^+ + y_1^-) \\
& f \geq (x_2^+ + x_2^-) + (y_2^+ + y_2^-) \\
& f \geq (x_3^+ + x_3^-) + (y_3^+ + y_3^-) \\
& x, y, x_1^+, x_1^-, y_1^+, y_1^-, x_2^+, x_2^-, y_2^+, y_2^-, x_3^+, x_3^-, y_3^+, y_3^- \geq 0
\end{aligned}$$

A quantity w (function or variable) that is unrestricted in sign can be modeled linearly by equating it to the difference of two nonnegative variables w^+ and w^- :

$$w = w^+ - w^-.$$

We have, in almost every situation,

$$w^+ w^- = 0.$$

In these cases we have

$$w^+ = \begin{cases} w, & \text{if } w \geq 0, \\ 0, & \text{otherwise,} \end{cases} \quad w^- = \begin{cases} -w, & \text{if } w < 0, \\ 0, & \text{otherwise.} \end{cases}$$

In addition, the minimax problem

$$\min \max \{f_1(\mathbf{x}), \dots, f_n(\mathbf{x})\}$$

of finding the variable values \mathbf{x} that minimize the maximum value of the linear functions $f_1(\mathbf{x}), \dots, f_n(\mathbf{x})$ can be modeled by first introducing a new variable λ , which will be the desired value. Since λ is to be a maximum value, we must ensure that, for each \mathbf{x} , we have, for each $k = 1, \dots, n$,

$$\lambda \geq f_k(\mathbf{x}).$$

To ensure that the maximum value is as small as possible, we wish to

minimize λ . Thus, our linear program would then be

$$\begin{aligned} \min \quad & \lambda \\ \text{s.t.} \quad & \lambda \geq f_k(\mathbf{x}), \quad k \in \{1, \dots, n\}. \end{aligned}$$

We can also write the maximin problem in such a general form (see Exercise 2.31).

2.8 VARIOUS FORMS OF LINEAR PROGRAMS

We have already seen that linear programs can be written using any combination of inequalities and equations. This is fine since we can transform any linear program into an equivalent one by changing the structure of any constraint, variable bound, or the objective function. These forms are equivalent in the sense that every optimal solution to the transformed problem can be directly translated back to an optimal solution to the original problem. In this section we discuss some of these transformations, while leaving others as exercises.

Changing Constraint Forms

In Section 2.1 we introduced the notions of slack and surplus variables. For example, consider the inequality

$$(2.10) \quad 3x_1 + 2x_2 + 6x_3 - 5x_4 \leq 9.$$

By introducing a slack variable $s = 9 - 3x_1 - 2x_2 - 6x_3 + 5x_4$, we can transform (2.10) into the equivalent form

$$(2.11) \quad 3x_1 + 2x_2 + 6x_3 - 5x_4 + s = 9,$$

where $s \geq 0$. Any feasible solution (x_1, x_2, x_3, x_4) for (2.10) generates an equivalent feasible solution (x_1, x_2, x_3, x_4, s) for (2.11), provided s is nonnegative. We can do the same thing for greater-than-or-equal-to constraints. For example, the constraint

$$(2.12) \quad 3x_1 + 2x_2 + 6x_3 - 5x_4 \geq 9$$

can be converted into the equivalent form

$$(2.13) \quad 3x_1 + 2x_2 + 6x_3 - 5x_4 - s = 9,$$

by using the surplus variable $s \geq 0$.

If constraint i is a less-than-or-equal-to (\leq) constraint

$$\sum_{j=1}^n a_{ij}x_j \leq b_i,$$

then it can be converted into an equality constraint (=) by adding a slack variable $s_i \geq 0$ to the left-hand side:

$$\sum_{j=1}^n a_{ij}x_j + s_i = b_i.$$

If constraint i is a greater-than-or-equal-to (\geq) constraint

$$\sum_{j=1}^n a_{ij}x_j \geq b_i,$$

then it can be converted into an equality constraint (=) by subtracting a (non-negative) surplus variable $s_i \geq 0$ from the left-hand side:

$$\sum_{j=1}^n a_{ij}x_j - s_i = b_i.$$

■ EXAMPLE 2.7

Consider the linear program

$$\max \quad 8x + 5y - 3z$$

s.t.

$$3x - 2y + 4z \leq 10$$

$$2x + 4y - z \geq 2$$

$$x, y, z \geq 0.$$

By adding a slack variable s_1 to the first constraint and a surplus variable s_2 to the second constraint, we can convert this linear program into the equivalent linear program

$$\begin{aligned}
\max \quad & 8x + 5y - 3z \\
\text{s.t.} \quad & \\
& 3x - 2y + 4z + s_1 = 10 \\
& 2x + 4y + z + s_2 = 2 \\
& x, y, z, s_1, s_2 \geq 0,
\end{aligned}$$

where each constraint is now an equality constraint.
For completeness, we should be able to go backward:

Rule: Given an equality constraint

$$\sum_{j=1}^n a_{ij}x_j = b_i,$$

we can convert it into the two inequalities

$$\begin{aligned}
\sum_{j=1}^n a_{ij}x_j &\leq b_i \\
\sum_{j=1}^n a_{ij}x_j &\geq b_i.
\end{aligned}$$

Changing Variable Bound Forms

Another transformation we need to consider is when a variable is not nonnegative but nonpositive or unrestricted in sign.

When a variable is nonpositive, it can be easily transformed into a formulation where the variable is nonnegative by using the “renaming” variable

$$x'_i = -x_i.$$

In this case, if $x'_i \leq 0$, then $x'_i \geq 0$.

■ EXAMPLE 2.8

Consider the following linear program:

$$\begin{aligned}
\max \quad & 8x + 5y - 3z \\
\text{s.t.} \quad & \\
& 3x - 2y + 4z = 10 \\
& 2x + 4y - z = 2 \\
& x, y \geq 0, \quad z \leq 0,
\end{aligned}$$

By renaming $z' = -z$, we get the equivalent

$$\begin{aligned} \max \quad & 8x + 5y + 3z' \\ \text{s.t.} \quad & 3x - 2y - 4z' = 10 \\ & 2x + 4y + z' = 2 \\ & x, y \geq 0, \quad z' \geq 0. \end{aligned}$$

But what if we have a variable that is unrestricted in sign, that is, can be either positive or negative. In this case, we use the “trick” first given in Section 2.7.

■ EXAMPLE 2.9

Consider the following linear program:

$$\begin{aligned} \max \quad & 8x + 5y - 3z \\ \text{s.t.} \quad & 3x - 2y + 4z = 10 \\ & 2x + 4y - z = 2 \\ & x, y \geq 0. \end{aligned}$$

Since z is unrestricted in sign, we can transform z into $z = z^+ - z^-$, where both $z^+, z^- \geq 0$. Doing so yields

$$\begin{aligned} \max \quad & 8x + 5y - 3(z^+ - z^-) \\ \text{s.t.} \quad & 3x - 2y + 4(z^+ - z^-) = 10 \\ & 2x + 4y - (z^+ - z^-) = 2 \\ & x, y, z^+, z^- \geq 0 \end{aligned}$$

or

$$\begin{aligned} \max \quad & 8x + 5y - 3z^+ + 3z^- \\ \text{s.t.} \quad & 3x - 2y + 4z^+ - 4z^- = 10 \\ & 2x + 4y - z^+ + z^- = 2 \\ & x, y, z^+, z^- \geq 0. \end{aligned}$$

Similar transformations can be made for variables with nonzero lower bounds (see Exercise 2.46) and to change the form of the objective between minimizing and maximizing (see Exercise 2.48).

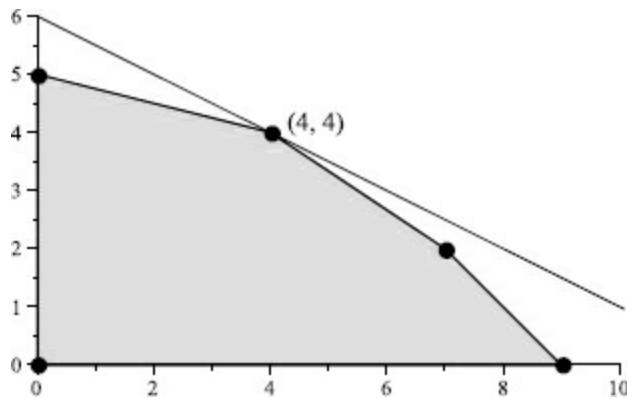
One last comment on the forms of linear programs is needed. At times there are constraints in our model whose removal from the problem does not alter the feasible region.

■ EXAMPLE 2.10

Consider the linear program

$$\begin{aligned} \max \quad & 8x + 3y \\ \text{s.t.} \quad & x + 4y \leq 20 \\ & x + y \leq 9 \\ & 2x + 3y \leq 20 \\ & x + 2y \leq 12 \\ & x, y \geq 0. \end{aligned}$$

FIGURE 2.1 Feasible region to Example 2.10.



Its feasible region is shown in [Figure 2.1](#). If we were to remove the constraint $x + 2y \leq 12$, then the feasible region does not change.

Redundant Constraint A constraint is *redundant* if its removal from or addition to the list of constraints to a mathematical program does not alter the feasible region.

It should be obvious that any redundant constraint can be removed from the mathematical program. However, this can be done only one constraint at a time, at which point all previously redundant constraints should be re-evaluated to see if they are still redundant (see Exercise 2.47).

Why worry about such transformations? In one respect, we now know that we can state our problem in different forms without much worry; for

example, if we are interested in the slack of a certain inequality constraint, we can add a slack variable to that constraint and output its optimal value when the problem is solved. Perhaps the most useful reason to transform our problem is to simplify our work later. In Chapter 8, we state an algorithm for solving linear programs with all equality constraints and nonnegative variables; by the equivalence of these transformations, this is the only form we need to (theoretically) consider.

2.9 NETWORK MODELS

Certain models have a special structure that allows them to be solved using specialized algorithms. One important class of such models is known as **Network Flow Models**, or just *Network Models*.

Typically, these models can be formulated on some graph-like, or *network*, structure. A *network* is defined by a set of points V known as *nodes* or *vertices* and ordered pairs of nodes A , known as *arcs*, that represent possible movement from one node to another. Often in network models, we are interested in the “flow” of material from one location (node) to another. For example, if we want to model the shipment of goods from a supplier to a customer, we could use a network where there is a node for each supplier, a node for each customer, and an arc from a supplier to a customer if that supplier can send material to that customer. We typically would not have an arc from a customer to a supplier in such a network, although this is not forbidden.

In network models, we often have material, be it utilities such as water or electricity, commercial goods such as food or other sellable items, or some abstract materials such as indicators telling the modeler what decisions to make, located at a subset of nodes. These nodes are referred to as *supply nodes*. Each supply node i has an amount of material s_i that needs to be moved to another set of nodes, known as *demand nodes*. Each demand node j , which could represent customers, for example, has demand d_j that it must receive from the supply nodes. There could be other nodes, known as *transshipment nodes*, that have neither supply nor demand, but through which goods may travel. We can think of these as warehouse locations, for example.

It is also possible for either an arc or a node to have a capacity associated with it, indicating how much material can pass to or through it. In addition, each arc can have a per-unit cost associated with it. Our variables are typically the amount sent over each arc; thus, there would be variables x_{ij} for each arc (i, j) .

We now examine some common network models: *transportation models*, *minimum cost network flow models*, and *shortest path models*.

Transportation Models

One common application of linear programming models is in the transportation industry, which delivers goods from one locale to another at minimum cost. A classical model from this industry is the **transportation problem**, in which all nodes are either supply nodes or demand nodes. An example of a transportation problem is given below.

■ EXAMPLE 2.11

A local baked goods company has two bakeries where they bake their goods, which they then ship to three different area stores to sell. Each bakery can produce up to 50 truckloads of baked goods per week, while the stores anticipate the following weekly demands, in number of truckloads.

Store	Demand
1	30
2	25
3	40

Each bakery can supply any of the stores, and the unit cost per truckload of shipping from a bakery to a given store is given in the table below.

	Store 1	Store 2	Store 3
Bakery 1	\$20	\$45	\$35
Bakery 2	\$35	\$35	\$50

How much should be sent from each bakery to each store so as to minimize the total shipping cost?

In this problem, the bakeries would be our supply nodes and our stores would be the demand nodes. We would have arcs (i, j) going from each bakery i to each store j , where $i = 1, 2$ and $j = 1, 2, 3$. The network is illustrated in [Figure](#)

2.2.

Let's define our decision variables as follows:

$$x_{ij} = \text{amount of truckloads of baked goods sent from bakery } i \text{ to store } j, \quad i = 1, 2, j = 1, 2, 3.$$

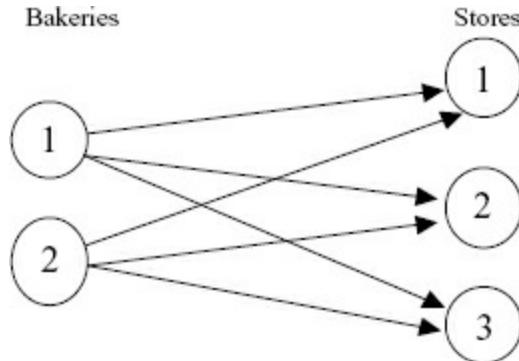
In a general transportation model, we would have variable x_{ij} representing the amount of flow from supply node i to demand node j . Using these variables, the number of truckloads sent from bakery i (supply node) to store j (demand node) is

$$\sum_{j=1}^3 x_{ij} = x_{i1} + x_{i2} + x_{i3}.$$

Since each bakery can send at most 50 truckloads out, we have the constraints

$$\sum_{j=1}^3 x_{ij} \leq 50, \quad i = 1, 2,$$

FIGURE 2.2 Transportation network example.



or $x_{11} + x_{12} + x_{13} \leq 50$ and $x_{21} + x_{22} + x_{23} \leq 50$. We next need to determine the constraints associated with the demand nodes. For each store j (demand node), the number of truckloads sent to store j is

$$\sum_{i=1}^2 x_{ij} = x_{1j} + x_{2j}.$$

Thus, given the demand that each store has, we'd have the constraints

$$\begin{aligned} x_{11} + x_{21} &\geq 30 \\ x_{12} + x_{22} &\geq 25 \\ x_{13} + x_{23} &\geq 40. \end{aligned}$$

Again, we need to send a nonnegative amount of flow across each arc, so

each $x_{ij} \geq 0$. Finally, the total cost associated with sending these trucks from bakeries to stores is

$$\text{cost} = 20x_{11} + 45x_{12} + 35x_{13} + 35x_{21} + 35x_{22} + 50x_{23},$$

giving us the linear program

$$\begin{aligned} \min \quad & 20x_{11} + 45x_{12} + 35x_{13} + 35x_{21} + 35x_{22} + 50x_{23} \\ \text{s.t.} \quad & \end{aligned}$$

$$x_{11} + x_{12} + x_{13} \leq 50$$

$$x_{21} + x_{22} + x_{23} \leq 50$$

$$x_{11} + x_{21} \geq 30$$

$$x_{12} + x_{22} \geq 25$$

$$x_{13} + x_{23} \geq 40$$

$$x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, 2, 3.$$

Did you notice that each variable corresponds to an arc in the network, while each constraint corresponds to a node? We'll come back to this notion when we study algorithms for solving this problem.

Solution An optimal solution to this problem is to send 30 truckloads from bakery 1 to store 1, 25 truckloads from bakery 2 to store 2, and 20 truckloads to store 3 from both bakery 1 and bakery 2, at a total cost of \$3175.

General Model In general, if there are m supply nodes, each with supplies s_i , and n demand nodes, each with demand d_j , and the cost associated with shipping items from supply node i to demand node j is c_{ij} , then the general transportation problem is given in Linear Program 2.9.

Linear Program 2.9 General Model for Transportation Problem

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

s.t.

$$\sum_{j=1}^n x_{ij} \leq s_i, \quad i \in \{1, \dots, m\}$$

$$\sum_{i=1}^m x_{ij} \geq d_j, \quad j \in \{1, \dots, n\}$$

$$x_{ij} \geq 0, \quad i \in \{1, \dots, m\}, \quad j \in \{1, \dots, n\}$$

Can we tell when there is going to be an optimal solution and when no

feasible solution exists? If the total supply is smaller than the total demand, we cannot possibly meet all the demand, so there is no feasible solution. In any other case, there is always a feasible solution to our problem. What happens if the total supply equals the total demand? It should be obvious that each of the constraints above must be equality constraints; fortunately, any transportation problem can be transformed into an equivalent transportation problem that has equal supply and demand, simply by adding an additional node and some additional arcs (see Exercises 2.44 and 2.45).

Minimum Cost Network Flow Models

The transportation problem described earlier in this section is very simplistic, in that there are no “middlemen” or warehouses that are used to hold goods before they arrive at the customers. When “middlemen” are present, we are faced with a different, but similar problem. These problems are referred to as **transshipment problems** in business and industry, or to a mathematician or computer scientist, **minimum cost network flow problems**.

■ EXAMPLE 2.12

PedalMetal Motors has two plants, two warehouses, and three customers. The locations of these are as follows:

Plants:	Detroit and Atlanta
Warehouses:	Denver and Cincinnati
Customers:	Los Angeles, Chicago, and Philadelphia

Cars are produced at plants, then shipped to warehouses, and finally shipped to customers. Detroit and Atlanta can produce 110 and 100 cars per week, respectively. Los Angeles requires 80 cars per week, Chicago, 70, and Philadelphia, 60. It costs \$10,000 to produce a car at each plant and the cost of shipping a car between two cities is given below.

From	To		From	To		
	Denver	Cincinnati		Los Angeles	Chicago	Philadelphia
Detroit	\$1253	\$637	Denver	\$1059	\$996	\$1691
Atlanta	\$1398	\$841	Cincinnati	\$2786	\$802	\$700

How can we meet PedalMetal Motors’s weekly demands at minimum cost?

In this network model, our supply nodes are the plants in Detroit and Atlanta, our demand nodes are the customers in Los Angeles, Chicago, and Philadelphia, and our transshipment (“middlemen”) nodes are the warehouses

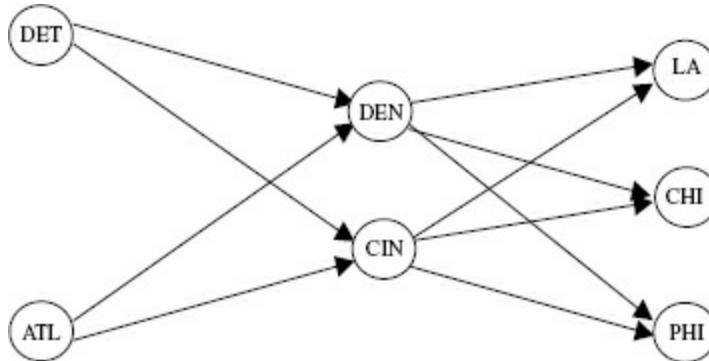
in Denver and Cincinnati. Note that, in this problem, we can send only cars from the plants (supply nodes) to the warehouses (transshipment nodes), and from the warehouses to the customers (demand nodes). A picture of this network is shown in [Figure 2.3](#).

Having this network, we again have the decision variables x_{ij} , where (i, j) is an arc in the network that represents the amount of goods shipped from node i to node j .

The constraints for the supply and demand nodes are similar to those found in the transportation problem, and we just list them below:

$$\begin{aligned}x_{DET,DEN} + x_{DET,CIN} &\leq 110 \quad (\text{Detroit}) \\x_{ATL,DEN} + x_{ATL,CIN} &\leq 100 \quad (\text{Atlanta}) \\x_{DEN,LA} + x_{CIN,LA} &\geq 80 \quad (\text{Los Angeles}) \\x_{DEN,CHI} + x_{CIN,CHI} &\geq 70 \quad (\text{Chicago}) \\x_{DEN,PHI} + x_{CIN,PHI} &\geq 60 \quad (\text{Philadelphia}).\end{aligned}$$

[FIGURE 2.3](#) Transshipment problem network.



The constraints associated with the warehouses are different. For these nodes, nothing is permanently kept there, so anything that is received is later sent to a customer. In other words, whatever enters a warehouse must leave it. This is typical of transshipment nodes. The constraints for these nodes, often called *balance constraints*, look like the following:

$$\text{flow in} = \text{flow out}.$$

For the warehouse in Denver, the number of cars entering the warehouse is

$$x_{DET,DEN} + x_{ATL,DEN},$$

while the number of cars being shipped to customers is

$$x_{DEN,LA} + x_{DEN,CHI} + x_{DEN,PHI},$$

giving us the balance constraint

$$x_{DET,DEN} + x_{ATL,DEN} = x_{DEN,LA} + x_{DEN,CHI} + x_{DEN,PHI}$$

For Cincinnati, we'd have the following constraint:

$$x_{DET,CIN} + x_{ATL,CIN} = x_{CIN,LA} + x_{CIN,CHI} + x_{CIN,PHI}$$

Our objective function is very similar to that of the transportation problem, where we have costs associated with shipping cars from one location to another. In addition, we have a cost associated with the actual production of cars. Since the number of cars produced is the number shipped from both plants, namely $x_{DET,DEN} + x_{DET,CIN} + x_{ATL,DEN} + x_{ATL,CIN}$, multiplying this sum by 10,000 gives us the cost of producing cars. The final linear program is given in Linear Program 2.10.

Solution The optimal solution calls for 20 cars to be shipped from Atlanta to Cincinnati, 80 cars from Atlanta to Denver, 110 cars from Detroit to Cincinnati, 70 cars from Cincinnati to Chicago, 80 cars from Denver to Los Angeles, and 60 cars from Cincinnati to Philadelphia; the total cost is \$2,481,590.

General Model When we discussed the transportation problem, we saw that if the total supply equals the total demand, then we can write the problem with only equality constraints. The same holds true for minimum cost network flow problems, and for the same reason the only way to meet the demands is if each supplier ships out all its supplies, which leads to demand being met exactly.

Linear Program 2.10 Model for Example 2.12

$$\begin{aligned}
\min \quad & 10000(x_{DET,DEN} + x_{DET,CIN} + x_{ATL,DEN} + x_{ATL,CIN}) \\
& + 1253x_{DET,DEN} + 637x_{DET,CIN} \\
& + 1398x_{ATL,DEN} + 841x_{ATL,CIN} \\
& + 1059x_{DEN,LA} + 996x_{DEN,CHI} + 1691x_{DEN,PHL} \\
& + 2786x_{CIN,LA} + 802x_{CIN,CHI} + 700x_{CIN,PHL} \\
\text{s.t.} \quad & \\
& x_{DET,DEN} + x_{DET,CIN} \leq 110 \\
& x_{ATL,DEN} + x_{ATL,CIN} \leq 100 \\
& x_{DET,DEN} + x_{ATL,DEN} = x_{DEN,LA} + x_{DEN,CHI} + x_{DEN,PHL} \\
& x_{DET,CIN} + x_{ATL,CIN} = x_{CIN,LA} + x_{CIN,CHI} + x_{CIN,PHL} \\
& x_{DEN,LA} + x_{CIN,LA} \geq 80 \\
& x_{DEN,CHI} + x_{CIN,CHI} \geq 70 \\
& x_{DEN,PHL} + x_{CIN,PHL} \geq 60 \\
& \text{all } x_{ij} \geq 0
\end{aligned}$$

Network Flow Model Let V denote the set of nodes and A denote the set of arcs in our network G . At each node $i \in V$, let s_i denote the supply at node i , and d_j the demand at node j , and define $b_i = s_i - d_i$; we shall assume that

$$\sum_{i \in V} b_i = 0,$$

that is, the total amount of supply in the network equals the total demand. The *Minimum Cost Network Flow Problem* is given by

$$\begin{aligned}
\min \quad & \sum_{(i,j) \in A} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j:(i,j) \in A} x_{ij} - \sum_{k:(k,i) \in A} x_{ki} = b_i, \quad i \in V \\
& x_{ij} \geq 0, \quad (i, j) \in A.
\end{aligned}
\tag{2.14}$$

Of course, the constraints associated with the demand nodes could be rewritten so that all the coefficients and right-hand sides are positive. Note that transportation models are just simplified versions of minimum cost network flow models, in which there are no transshipment nodes.

Importance of Network Models

Now that we have seen various network models, a reasonable question to ask is “Why do we separate them as a special class of linear program?” First, all the models considered in this section are various forms of the minimum cost network flow model. In fact, this can be made into a more general statement.

Claim 2.1 *Nearly all network models can be formulated as minimum cost network flow problems.*

However, the importance of network models lies in their solution. Unlike general linear program where we cannot expect the variables to have integer values, network models have this nice feature.

Claim 2.2 *Any problem modeled as a minimum cost network flow problem has an optimal solution that has only integral values, provided each supply and each demand has integral value.*

While we will not prove this claim here, it is important to note its implication. Given integral data and a network structure, we can guarantee the existence of an integer-valued optimal solution (provided one exists). For all these network models, as well as others not described here, a solution is found not by solving the linear program described in this section but through an algorithm that specifically utilizes the inherent mathematical structure found in the network. Such combinatorial algorithms are typically more efficient in practice than those solving linear programs, and as such are of interest to researchers and practitioners. We will explore some of these algorithms in Chapter 12.

Shortest Path Models

Not every network problem deals with the shipment of goods from one locale to another. Often, it is worthwhile to use a network model to help make consecutive decisions. In this case, the nodes represent starting/ending points in time, arcs represent some activity to be performed, and we are interested in “paths” through the network that help us determine which activities to perform. One important application of this idea is in equipment replacement, as described in the following example.

■ EXAMPLE 2.13

At a local college, the Print Shop provides photocopying services for the entire school, often printing large quantities of exams and worksheets for professors to give to students. Due to the amount of copies made each school year, the college is interested in determining when to purchase a new high-speed copier over the next 6 years. During those years that a copier is not purchased, maintenance must be performed to ensure that the machine is not down for any extended period. Based on the copier company, the maintenance cost for a machine depends on its age. The table below is the company's estimated maintenance cost per age of machine.

Age at Beginning of Year	Maintenance Cost for Next Year
0	\$2000
1	\$3500
2	\$6000
3	\$9500
4	\$14,000

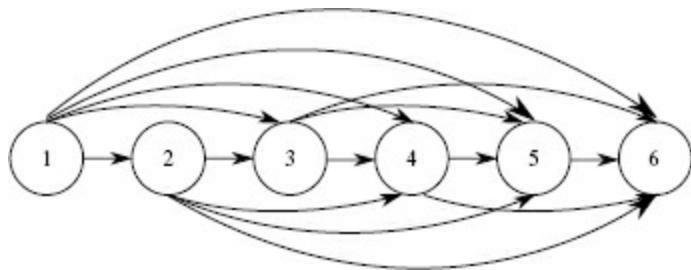
Suppose that, at the beginning of this school year, a new copier has been purchased, at a cost of \$10,000. The cost (in today's dollars) of purchasing a machine at the beginning of each of the next 5 years is given below.

Year	Purchase Cost
1	\$10,000
2	\$13,000
3	\$16,500
4	\$20,000
5	\$25,000

There is no trade-in value when a machine is replaced. Determine the years in which a new copier should be purchased in order to minimize the total (purchase plus maintenance) cost of having a machine for 5 years.

Even though we are not shipping any goods here this problem still has a network structure. Let node i represent the beginning of year i ; since we want to examine the costs over 5 full years, we need six nodes. Each arc (i, j) represents the cost incurred by purchasing a new copier at the beginning of year i , and maintaining it until the beginning of year j , at which time a new machine will be bought. The arcs in this problem are of the sort (i, j) , where $i < j$. Hence, there will be $5 + 4 + 3 + 2 + 1 = 15$ arcs. This network is given in [Figure 2.4](#).

FIGURE 2.4 Shortest path problem network for Example 2.13.



How does this solve the problem? It all depends on the concept of a *path*.

Path In a network, a *path* is an ordered sequence of arcs (i, j) such that any node i is “visited” at most once.

In our problem we are interested in a path from node 1 (start of year 1) to node 6 (end of year 5/beginning of year 6). Each arc in the path will indicate if we are to purchase a new machine. For example, if we have the path $(1, 2)$, $(2, 4)$, $(4, 6)$, then we will purchase a new machine at the beginning of years 1, 2, and 4. Note that, since we are interested only in a 5-year plan, we do not know what we’ll do at the beginning of year 6.

How can we identify such a path? Think of the problem as a minimum cost network flow problem where we have 1 unit of supply at node 1, 1 unit of demand at node 6, and the other nodes are transshipment nodes. Because all supply and demand values are integers, the optimal solution to our problem will be integer-valued as well, and in this case the values will be either 0 or 1. Those variables with value 1 indicate the arcs on the minimum cost path from node 1 to node 6. Hence, we have used the structural properties of network flow problems to aid in our model creation.

What about the costs? Each arc has a cost corresponding to the total cost accumulated by purchasing the machine at the beginning of year i and maintaining it up through year $j - 1$. Also, if there were a resale value on the machine, we would have to include that as well. Generally, in these types of problems, we calculate the cost on arc (i, j) as

$$\begin{aligned} c_{ij} = & (\text{purchase price in year } i) \\ & + (\text{maintenance costs during years } i, i+1, \dots, j-1) \\ & - (\text{resale price in year } j \text{ after } j-i \text{ years of service}). \end{aligned}$$

For example, if we purchase a new machine at the beginning of year 2, then

maintain it until the beginning of year 5 when we purchase a new copier, the cost on arc (2, 5) would be (in thousands of dollars)

$$c_{25} = 13 + 2 + 3.5 + 6 = 24.5.$$

The cost (in thousands of dollars) matrix for the remaining arcs would be

$$c = \begin{bmatrix} - & 12 & 15.5 & 21.5 & 30.5 & 43.5 \\ - & - & 15 & 18.5 & 24.5 & 34 \\ - & - & - & 18.5 & 22 & 28 \\ - & - & - & - & 24 & 27.5 \\ - & - & - & - & - & 27 \end{bmatrix}.$$

Note that the total supply equals the total demand, so our linear program is given in Linear Program 2.11. This problem generally falls under the name *Shortest Path Problems*, since we are interested in finding the shortest path from one location to another. We can generalize this approach to finding the shortest path from one node to all other nodes (see Exercises 2.42 and 2.43).

Linear Program 2.11 Model for Shortest Path Example 2.13

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

s.t.

$$x_{12} + x_{13} + x_{14} + x_{15} + x_{16} = 1$$

$$x_{12} - x_{23} - x_{24} - x_{25} - x_{26} = 0$$

$$x_{13} + x_{23} - x_{34} - x_{35} - x_{36} = 0$$

$$x_{14} + x_{24} + x_{34} - x_{45} - x_{46} = 0$$

$$x_{15} + x_{25} + x_{35} + x_{45} - x_{56} = 0$$

$$x_{16} + x_{26} + x_{36} + x_{46} + x_{56} = 1$$

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

Solution Solving Linear Program 2.11 results in the solution $x_{13} = x_{36} = 1$ and all other variables equal to 0. Thus, the optimal maintenance plan is to purchase new machines at the beginning of years 1 and 3 only, resulting in a cost of \$43,500.

Multicommodity Flows There are various ways we can generalize the notion of a network flow problem. One useful idea is to allow multiple classes of flow to travel on the same network. For example, we could have a distribution network in which there are different commodities being shipped

from suppliers to customers. In such cases, the total amount shipped between any two locations must satisfy a single upper bound. This problem is referred to as the *multicommodity flow problem*.

To model this problem, we extend the definition of a flow variable to x_{ij}^k , which gives the amount of commodity k sent along arc (i, j) in the network. If K represents the set of different commodities, we would also need to know the amount b_i^k of commodity k at each node i and any bounds u_{ij}^k on the amount of commodity k sent on arc (i, j) . As currently formulated, this problem could be decomposed into k separate problems, one for each commodity. However, we often combine these subproblems onto one network by also requiring that the total amount sent on arc (i, j) is no more than u_{ij} , that is,

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}.$$

Multicommodity Flow Problem The *multicommodity flow problem* is a generalization of the traditional network flow problem, where there are now k different commodities being simultaneously sent on a network $G = (V, A)$. This problem is formulated as

$$\begin{aligned}
 & \min \sum_{k \in K} \sum_{(i, j) \in A} c_{ij}^k x_{ij}^k \\
 & \text{s.t.} \\
 & \quad \sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(j,i) \in A} x_{ji}^k = b_i^k, \quad i \in V, k \in K \\
 & \quad \sum_{k \in K} x_{ij}^k \leq u_{ij}, \quad (i, j) \in A \\
 & \quad 0 \leq x_{ij}^k \leq u_{ij}^k, \quad (i, j) \in A, k \in K.
 \end{aligned} \tag{2.15}$$

It should be noted that, unlike the traditional network flow problem, **the integrality of the variables x_{ij}^k cannot be guaranteed, even if all problem data are integral**. For more information on this and other properties of the problem, please check out Ahuja et al. [2].

Summary

We've seen various linear optimization modeling techniques that are common in the formulation of linear programming models. Some models, such as blending constraints and network flow problems, appear regularly in real-world situations. We should note that there were times when integer values would have been appropriate, but the model did not necessarily ask for them. In the next chapter, we will consider models where integer variables play a prominent role in the models we construct.

EXERCISES

For each of the problems provide the formulated optimization model and its optimal solution and value.

2.1 CT Furniture Company makes wooden picnic tables and chairs to sell to various stores. These stores request both finished and unfinished furniture. To make this furniture, CT must purchase the wood by the board foot. It takes 25 board feet to make one table and 10 board feet to make one chair. Each board foot must be purchased for \$2, and up to 10,000 board feet can be purchased. It takes 4 hours to make an unfinished table and 2 hours to make an unfinished chair. To finish an unfinished table (by priming, sealing, and painting it), it takes an additional 8 hours, while it takes 6 hours to finish an unfinished chair. A total of 2500 hours are available, which can be proportioned as necessary. We can assume that all pieces of furniture produced can be sold at the following prices: \$80 for unfinished table, \$120 for finished table, \$30 for unfinished chair, and \$55 for finished chair. In addition, due to typical demand, we know that, for every table produced, at least 2 chairs must be produced. If we have special orders that must be filled, calling for at least 200 tables and 450 chairs to be sold, determine how much of each type of furniture to produce in order to maximize CT's profit.

2.2 In Example 2.1, suppose we wish to analyze the effect of having more labor hours available for production. Resolve the problem for each of the following labor amounts: 290, 300, 310, and 320. How does the optimal profit and the production amounts change as we increase total labor? Describe how the optimal solution changes as the labor amounts change,

including why you believe the production amounts change as much as they do.

2.3 A film packaging plant can manufacture four different thicknesses (1, 3, 5, and 0.5 mm) in any combination. Each thickness requires time on each of three machines in minutes per square yard of film, as shown in the table below. Each machine is available 60 hours per week. The table also gives revenue and cost per square yard for each thickness. Variable labor costs are \$25 per hour for machines 1 and 2, and \$35 per hour for machine 3. Formulate and solve a profit-maximizing LP model for this problem, given the maximum demands for each thickness.

Thickness	Time (min)			Max Demand	Revenue	Cost
	1	2	3			
1mm	5	8	9	400	\$110	\$30
3mm	4	7	5	250	\$90	\$10
5mm	4	5	4	200	\$60	\$10
0.5mm	6	10	6	450	\$100	\$20

2.4 Wood Built Bookshelves (WBB) is a small wood shop that produces three types of bookshelves: models A, B, and C. Each bookshelf requires a certain amount of time for cutting each component, assembling them, and then staining them. WBB also sells unstained versions of each model. The times, in hours, for each phase of construction and the profit margins for each model, as well as the amount of time available in each department over the next 2 weeks, are given below.

Model	Labor (h)			Profit Margin	
	Cutting	Assembling	Staining	Stained	Unstained
A	1	4	7	\$60	\$30
B	0.5	3	5	\$40	\$20
C	2	6	8	\$75	\$40
Labor available	200	700	550		

Since this is the holiday season, WBF can sell every unit that it makes. Also, due to the previous year's demand, at least 20 model B bookshelves (stained or unstained) must be produced. Finally, to increase sales of the stained bookshelves, at most 50 unstained models are to be produced.

(a) Formulate and solve a linear program to determine what WBF should produce in order to maximize its holiday profits?

- (b)** In part (a), how many extra hours are available in each phase of construction?
- (c)** Resolve the model after changing the profit margin for the stained model A bookshelf to values 50, 55, 65, and 70. How does the solution change for these values?
- (d)** Resolve the model after changing the profit margin for the unstained model A bookshelf to values 20, 25, 35, and 40. How does the solution change for these values?
- (e)** Resolve the model after changing the available cutting hours to 150, 175, 225, and 250. How does the solution and optimal value change for these values?
- (f)** Resolve the model after changing the available assembling hours to 650, 675, 725, and 750. How does the solution and optimal value change for these values?

2.5 Sycamore Basketball Company forecasts a 500-unit demand for its latest outdoor basketball hoop system during the next quarter. This hoop is assembled from three major components: support pole, backboard, and rim. Below are the production times and labor hours available. Note that each component produced must go through each department.

Department	Production Times (h)			Time Available (h)
	Pole	Backboard	Rim	
A	2	2.5	1	2000
B	0.5	1	1.5	900
C	1	2	1	1500

Until now, Sycamore Basketball has manufactured all its components. However, it has never had such demand, and is unsure if it can produce all 500. Thus, management has allowed for contracting the production of support poles or rims to a local firm. Below are the estimated costs for both manufacturing and purchasing each component.

Component	Manufacturing Cost	Purchase Cost
Pole	\$60	\$95
Backboard	\$80	—
Rim	\$30	\$45

- (a)** Formulate and solve a linear program to help determine the make-or-buy policy for each component if Sycamore Basketball wants to minimize costs.
- (b)** In part (a), which department has unused hours in the optimal solution?
- (c)** Resolve the problem for the following manufacturing costs on poles: 50, 55, 65, and 70. How does the optimal solution and value change?
- (d)** Resolve the problem for the following purchase costs on rims: 40, 50, 55, and 60. How does the optimal solution and value change?
- (e)** Resolve the model after changing the available hours in department C to 1350, 1375, 1400, 1450, and 1475. How does the solution and optimal value change for these values?

2.6 National Disc Corp. produces the discs used in producing DVDs and Blu-Ray discs. Their local plant runs 24 hours a day, 7 days a week. In a given day, there are requirements for the total number of employees that must be at the plant. These are given below.

Hours	Employees Needed
12 AM–4 AM	8
4 AM–8 AM	10
8 AM–12 PM	16
12 PM–4 PM	21
4 PM–8 PM	18
8 PM–12 AM	12

Employees can either work 8-hour or 12-hour shifts, starting at the times stated above (12-hour shifts can start only at 12 AM/PM or 8 AM/PM). Those working 8-hour shifts cost the company \$40 per hour in benefits, and those working 12-hour shifts cost the company \$60 per hour. Determine how National should staff the local plant so as to minimize labor costs.

2.7 In Exercise 2.6, we planned a schedule for only one day of the week. To plan the schedule for the entire week, we're told that National requires those working 8-hour shifts to work 5 days straight, with 2 days off, while those working 12-hour shifts work 3 consecutive days, followed by 4 days off. If at most $\frac{1}{3}$ of its employees can work 12-hour shifts, how should

National schedule its employees in order to minimize labor costs?

2.8 In Exercise 2.7, suppose now that those working 12-hour shifts work 3 consecutive days, followed by 3 days off, then 2 days at work, followed again by 2 off days, then 2 days working, then 2 days off. How should National schedule its employees now? Assume again that at most $\frac{1}{3}$ of its employees can work 12-hour shifts.

2.9 A company produces three products from raw material and the other products. Each pound of raw material undergoes processing and yields 3 ounces of product 1 and 1 ounce of product 2. Each pound of raw material costs \$25 to purchase and takes 2 hours of labor to process. Each ounce of product 1 can be handled in one of the three ways. First, it can be sold for \$10 per ounce. Second, it can be processed into $\frac{2}{3}$ ounce of product 2. This requires 2 hours of labor and costs an additional \$1. Third, it can be processed into $\frac{1}{2}$ ounce of product 3. This requires 3 hours of labor and costs an additional \$2. Each ounce of product 2 can be used in one of the two ways. First, it can be sold for \$20 per ounce. Second, it can be processed into $\frac{3}{4}$ ounce of product 3. This requires 1 hour of labor and costs \$6 more. Product 3 is sold for \$30 per ounce. The maximum numbers of ounces of products 1, 2, and 3 that can be sold are, respectively, 5000, 4000, and 3000. A maximum of 25,000 hours of labor are available. Formulate and solve a linear program that allows the company to maximize their profit?

2.10 MakeIt Company produces three products from raw material. Thirty thousand pounds of raw material is available. Each pound of raw material can be transformed into 0.4 pounds of product 1, 0.3 pounds of product 2, and 0.2 pounds of product 3, while 0.1 pounds is lost as waste material. In addition, each pound of product 1 can be used to produce 0.6 pounds of product 2, 0.3 pounds of product 3, and 0.1 pounds of waste material; this pound of product 1 cannot be sold later. Assuming that there are enough available resources to make the necessary amount of each product, formulate and solve a linear program that allows MakeIt to sell at least 4000 pounds of product 1, 8000 pounds of product 2, and 10,000 pounds of product 3 while minimizing the total amount of waste material produced?

2.11 Cavity Candies mass produces an assortment of hard candies, consisting of three flavors: cinnamon, root beer, and grape. Bag 1 must have between 75 and 100 pieces of candy, with at least 40% being cinnamon, at least 30% being grape, and between 15% and 30% being root beer. Bag 2 must have between 50 and 75 candies, with no more than 30% cinnamon, at least 40% grape, and at least 20% root beer. Each bag can contain at most 60 cinnamon, 75 grape, and 75 root beer candies at costs of \$0.01, \$0.02, and \$0.015 per candy, respectively. If bag 1 generates \$0.03 per piece of candy in revenue, and bag 2 generates \$0.025 per piece in revenue, formulate and solve a linear program to help maximize Cavity's per-bag profits?

2.12 Mammoth Oil manufacturers three types of gasoline (gas 1, gas 2, and gas 3). Each type is produced by blending three types of crude oil (crude 1, crude 2, and crude 3). The purchase prices per barrel of crude oil are given below. Mammoth can purchase up to 5000 barrels of each type of crude oil daily. Each type of oil has associated with it two numbers, the octane rating, which is a measure of “engine knocking,” and an overall quality rating, which encompasses many other measures. Below are the values of each rating for the three types of crude oil available, as well as their per-barrel purchase price.

	Purchase Price/Barrel	Octane Rating	Quality Rating
Crude 1	\$55	85	50
Crude 2	\$65	90	65
Crude 3	\$75	94	85

The three types of gasoline differ in their octane rating. The crude oil blended to form gas 1 must have an average octane rating of at least 87 and quality rating at least 60, the oil blended to form gas 2 must have an average octane rating of at least 89 and quality rating 70, and the oil blended to form gas 3 must have an average octane rating of at least 91 and quality rating 80. The octane ratings of the three types of oil are given above. It costs \$4 to transform one barrel of oil into one barrel of gasoline, and Mammoth's refinery can produce up to 14,000 barrels of gasoline daily. Mammoth's customers require the following amounts of each gasoline daily: gas 1, 4000 barrels; gas 2, 3000 barrels; and gas 3, 2000 barrels. The company considers it an obligation to meet these demands.

Formulate and solve a linear program that will enable Mammoth to minimize the cost of meeting customer demand.

2.13 At Midwest Steel, steel is produced by combining various alloys and scrap pieces of steel in high-temperature furnaces. For a 100-ton piece of steel, three different alloys and two different types of scrap material are to be used. Each alloy and scrap piece have different properties and chemical makeups. The various characteristics of each alloy and scrap are given below, as well as their availability and cost per ton.

	Characteristics (per ton)					
	Carbon	Nickel	Chromium	Tensile Strength	Availability	Cost
Alloy 1	1.75%	2.0%	3.5%	60,000	50	\$150
Alloy 2	2.45%	3.0%	0.8%	40,000	50	\$120
Alloy 3	2.80%	4.0%	1.2%	90,000	20	\$80
Scrap 1	3.10%	4.5%	3.9%	120,000	30	\$35
Scrap 2	3.50%	5.5%	2.8%	70,000	40	\$20

The customer wants this produced steel to meet the following chemical requirements: the carbon composition must be between 2% and 3%, the nickel composition cannot exceed 4%, the chromium composition must be between 1.3% and 2.7%, and its tensile strength must be between 50,000 and 80,000 psi. Assuming that tensile strength of the produced steel is a weighted average of the tensile strength of each alloy and scrap piece, determine how Midwest can produce this 100-ton piece of steel at minimum cost.

2.14 Cattle feed can be mixed from oats, corn, and alfalfa. Farmers want to construct a feed that meets all recommended daily allowances (RDA) of protein, fat, and fiber. Below is the current cost per ton (in dollars) of each type of feed, as well as the percentage of RDA for each ingredient.

	% Protein	% Fat	% Fiber	Cost/Ton
Oats	13	22	40	\$80
Corn	7	10	30	\$110
Alfalfa	4	15	60	\$90

Formulate and solve a linear program that will obtain the minimum cost way to produce 1 ton of feed by combining oats, corn, and alfalfa such that the mixture contains at least 8% of the RDA for protein, between 12% and

16% of the RDA for fat, and no more than 50% of the RDA for fiber.

2.15 In Example 2.3, we saw that the optimal solution produced more regular gasoline than was required. Suppose we allowed HoosCo to produce up to 16,000 barrels of regular gasoline without penalty, but penalized the company \$100 for every additional barrel over 16,000. How can we model this using linear programming, and what is an optimal production for this situation?

2.16 (Based on Bean et al. [10]) GoShop Development is building a new shopping plaza and is trying to allocate retail space among various stores. The plaza will include women's clothing stores, children's clothing stores, shoe stores, electronic stores, jewelry stores, and book stores. To simplify its communications, all stores in each class are owned by a single company; for example, all jewelry stores are owned by one large corporation, even though the stores are named differently. Each of the six store classes have minimum and maximum space requirements (in square feet) and have provided GoShop with estimates on the per square-foot profit margins for GoShop. GoShop has also estimated the per square-foot cost of building each class of shop, since they have different physical requirements. These values are provided in the table below.

Store	Min Space	Max Space	Cost	Profit
Women's	1500	6500	120	50
Children's	750	5000	100	30
Shoe	800	3000	80	70
Electronic	1000	4000	150	50
Jewelry	1000	5000	75	90
Book	1000	4000	90	60

The shopping plaza will encompass 15,000 square feet of retail space. In addition, GoShop would like to ensure that at least 45% of the space is given to clothing stores, while at most 15% is provided to book stores. If GoShop has a construction budget of \$25 million dollars, formulate and solve a linear program that tells GoShop how to allocate space to the various store classes in order to satisfy the various requirements and maximize its profit margins.

2.17 In Example 2.3, resolve the problem for each of the following sets of parameter changes. In each case indicate how both the optimal solution

and its value have changed.

- (a) Cost per barrel of type 2 oil is \$25, \$30, \$40, and \$50.
- (b) Maximum availability of type 1 oil is 7000, 8000, 9000, and 11,000.
- (c) Minimum percentage of type 1 oil in regular gasoline is 15%, 20%, 30%, and 40%.
- (d) Maximum percentage of type 3 oil in premium gasoline is 10%, 15%, 25%, and 30%.
- (e) Average octane rating of regular gasoline is 87.5, 88, 88.5, 89.5, and 90.
- (f) Sale price per barrel of regular gasoline is \$55, \$60, \$70, and \$75.

2.18 Suppose in Example 2.5 we cannot hold more than 150 of either type of engine in inventory at the same time. How does this affect the optimal solution and value of the model?

2.19 (Based upon Carino and LeNoir [22]) WellBuilt Cabinet Co. owns a cabinet manufacturing facility that not only makes cabinets but also produces the wood panels or “blanks” used. The wood used to produce blanks come from either logs that are milled and dried within the company or lumber boards that are purchased either dried or green (undried). All undried wood is dried in the company’s kilns. The logs can be purchased in any of five diameters, each producing different amounts of blanks. The lumber boards can be purchased in one of the two grades. Pricing, yield, and availability for all log diameters and lumber grades are given in the tables below.

Log Diameter	Cost/Log	Blanks/Log	Max Logs
8	100	70	100
10	120	90	75
14	150	120	60
18	175	150	40

Lumber Grade	Cost/Board Foot		Blanks/Board Foot	Max Board Feet
	Dried	Green		
1	1.85	1.60	0.15	4000
2	1.15	1.00	0.06	10,000

WellBuilt needs to produce at least 2000 blanks per week, but its mill can

handle at most only 1000 logs and its kiln can dry at most 30,000 board feet of lumber. Formulate and solve a linear program that allows WellBuilt to meet its demands while minimizing cost.

2.20 During the next 4 months your company must meet (on time) the following demands for plastic garbage cans: 3000 large and 2500 small in month 1; 4500 large and 4000 small in month 2; 3000 large and 4000 small in month 3; and 4000 large and 4000 small in month 4. At the beginning of month 1, 75 large and 50 small cans are on hand. Each can is produced from plastic bought from another firm and must go through two different areas before completion. Each month, you are automatically shipped 30,000 pounds of plastic to make the cans. Those cans not sold in a given month must be stored in a local storage facility. Below are the requirements for producing and storing each can, in addition to its production cost per unit.

Can	Plastic (lb)	Time (h)			Storage Units	Costs	
		Machine	Painting			Production	Storage
Large	5	0.1	0.05		6	\$15	\$2
Small	3	0.08	0.04		3	\$10	\$1

There are 650 machine hours and 350 painting hours available each month, as well as 10,000 units of storage available. Production in a given month can be used to meet that month's demand. Formulate and solve a linear program to determine the optimal production schedule.

2.21 In Exercise 2.20, suppose that you can purchase additional plastic at a cost of \$4 per pound. How does this change your production schedule?

2.22 Fitness Sneaker Company manufactures and sells (you guessed it) sneakers. During each of the next 6 months, it forecasts the following demands (pairs of sneakers).

Month	1	2	3	4	5	6
Demand	6000	5000	8000	4000	7000	5000

All demand must be met during that month. Each sneaker is made by workers and requires 20 minutes per pair. Each worker works 200 hours per month and can work up to 40 hours of overtime per month. Workers are paid a salary of \$3000 per month, plus \$75 per hour of overtime. Prior

to each month's production, Fitness Sneaker can either hire additional workers or lay off some of its current workers. Due to administrative and other expenses, it costs \$2000 to hire a worker and \$3000 to fire a worker. Currently, at most 3000 pairs of sneakers can be stored in inventory, and this number is calculated at the end of the month (after all production). Each stored pair costs \$5 per month in storage fees. If there are 15 workers and 1000 pairs of sneakers in storage at the beginning of month 1, determine how Fitness Sneaker can minimize its cost of meeting the demand.

2.23 In Exercise 2.22, suppose that the demand for each month need not be met during that month. For example, the demand for month 1 need not all be met in month 1. However, the demand in months 4 and 6 must be met during those months. How should Fitness Sneaker schedule its sneaker manufacturing?

2.24 Quality Cabinets produces various cabinets that house entertainment electronics, including televisions, video game equipment, DVRS, and stereo equipment. They manufacture and sell three models, based on the size of the flat-screen televisions: standard (for 32-inch TVs), deluxe (for 40-inch TVs), and enhanced (for 50-inch TVs). To produce these cabinets, various pieces of wood are custom-sized, assembled to form the cabinets. These structures are then finished and painted in separate departments. Below are the times (in hours) required in each department for the various models, as well as the amount of wood planks needed for each product and their unit profit margins.

	Assembly	Finishing	Painting	Wood Required	Profit Margin
Standard	2	3	2	8	\$25
Deluxe	3	3.5	4	12	\$45
Enhanced	4	4	5	17	\$60

In a typical week there is 250, 300, and 400 hours, respectively, available in each department. Unfortunately, this week there are only 5000 wood planks available from their local supplier. In addition, they have orders for at least 100 standard, 75 deluxe, and 40 enhanced cabinets that must be met this week. To meet this demand, management has authorized the purchase, from a supplier, of assembled cabinets; however, only standard and deluxe sizes can be purchased. These cabinets only need to be finished

and painted. They take one hour less in the finishing department than those assembled within the company, and take the same amount of painting time. The profit margins for these pre-assembled models are now \$10 for standard and \$20 for deluxe models. How many of each model should be built and/or purchased so as to meet demand and maximize profit?

2.25 You have been asked to determine how long a project will take to be completed. This project consists of 11 tasks of varying time durations (measured in days) that must all be completed. Unfortunately, not all tasks can be done at the same time (although some can be done in parallel); certain predecessor tasks must be completed before others can be started. The table below gives the duration and predecessors for each task. Thus, for example, task J cannot begin before each of tasks F, G, and H are completed. How quickly can this entire project be completed?

Task	Duration	Immediate Predecessors
A	2	None
B	3	A
C	5	B
D	7	B
E	3	B
F	4	D
G	2	E
H	6	E
I	8	C, F
J	6	F, G, H
K	9	I, J

2.26 Now suppose that, in Exercise 2.25, we have the opportunity to reduce the time duration of any task if we are willing to invest some money. The chart below gives the maximum number of days each task can be reduced and how much it costs per day to reduce the duration. How much would it cost to reduce the total project time to 28 days?

	Task										
	A	B	C	D	E	F	G	H	I	J	K
Max Days	0	1	2	3	1	1	0	3	3	3	4
Cost/day	—	100	150	200	150	100	0	250	200	175	200

2.27 Specialty Containers produces plastic containers for welding rods. They have been producing the containers in a converted warehouse. Because of this, some operation areas, such as shipping, are set in certain locations; other areas are capable of moving. In order to increase

production, a new casting machine is being bought to replace the old one. Management feels that it is now time to redesign the work area to make production more efficient. To do this, the location of the casting machine and storage area are being moved. These facilities interact with each other and with three existing operations: where the raw plastic is stored, finishing area, where the containers are prepared for the customers, and the shipping gate, where finished containers are processed out of the plant. A coordinate system quantifies the locations of all three existing facilities. The following table displays material handling costs of expected traffic between facilities.

Material Handling Cost/Foot	Container Casting	Container Storage
Container storage	\$5.00	–
Plastic storage	\$2.15	–
Finishing area	\$0.75	\$0.85
Shipping	–	\$0.50

Currently, the plastic storage area is located at position (0, 75), the finishing area is at (25, 25), and the shipping area is at (100, 0). If all transportation of material must be parallel to the x - and y -axes, where should we put the new facilities to minimize total material handling cost?

2.28 Suppose in Exercise 2.27 we wish to minimize the largest material handling cost. Where should the new facilities now be located?

2.29 During a recent solar car race, you gathered the following data on daily power consumption and total mileage traveled.

Power consumed	10	8	13	15	9
Distance	60	55	75	81	62

There is believed to be a linear relationship between power and distance, that is, of the form

$$\text{distance} = m(\text{power}) + b,$$

for some unknown values of m and b , along with some random error term. In a statistics class, you would generate a least-squares regression line that minimizes the sum of squared difference between the observed distances and the predicted distances, based on the linear model, that is,

$$\min \sum_{j=1}^m (\text{distance}_k - (m\text{power}_k + b))^2.$$

Suppose instead you want to minimize the sum of the absolute deviations between the observed distances and predicted distances, that is,

$$\min \sum_{j=1}^m |\text{distance}_k - (m\text{power}_k + b)|.$$

What values of m and b will give this? Note that both m and b should be nonnegative.

2.30 In Exercise 2.29, suppose you wanted to minimize the maximum deviation between observed and predicted distances. What values of m and b will give this?

2.31 In Section 2.7 we saw how to formulate the minimax problem

$$\min_{\mathbf{x}} \max \{f_1(\mathbf{x}), \dots, f_n(\mathbf{x})\}$$

as a linear program (provided each function f_k is linear in \mathbf{x}). How can we write the maximin problem

$$\max_{\mathbf{x}} \min \{f_1(\mathbf{x}), \dots, f_n(\mathbf{x})\}$$

as a linear program?

2.32 Indiana Power is a local cooperative that supplies the power needs of four cities using three power plants. The potential supply from each power plant and the peak power demand for each city (each given in millions of kilowatt-hours of electricity) are given below.

	Supply
Plant 1	85
Plant 2	115
Plant 3	100

	Demand
City 1	115
City 2	70
City 3	65
City 4	50

Finally, the cost (in dollars) of sending a million kWh from each plant to

each city is given below.

	City 1	City 2	City 3	City 4
Plant 1	10	7	10	6
Plant 2	7	12	16	9
Plant 3	12	8	13	7

Indiana Power wants to find the lowest cost method for meeting demand of the four cities. Formulate and solve a transportation problem that minimizes the cost to Indiana Power.

2.33 Suppose that, in Exercise 2.32, city 4 finds that it needs an additional 25 million kWh at its peak; thus, we cannot satisfy every city's demand with our current supply amounts. Since it cannot immediately increase its output, Indiana Power wants to satisfy its demand as much as possible while still minimizing its total cost. To do so, it can purchase additional electricity from another cooperative at the following shipping costs:

City	1	2	3	4
Cost	50	40	60	70

Formulate and solve a transportation problem that minimizes the total cost to Indiana Power.

2.34 Four print jobs need to be completed at a university print shop, and five machines are available for assignment to these jobs. Each machine can be assigned to at most one job. The time required (in minutes) for each machine to complete each job is given below. Formulate and solve a linear program or network model to determine an assignment of machines to jobs that minimizes the time needed to complete the jobs.

	Job 1	Job 2	Job 3	Job 4
Machine 1	2	5	8	9
Machine 2	8	12	6	5
Machine 3	5	8	11	7
Machine 4	7	4	6	10
Machine 5	10	5	4	7

2.35 (Based on Hansen and Wendoll [55]) During the month of December you must make four round-trips between Indianapolis and Boston. The dates for the trips are given below.

Leave Indianapolis

Leave Boston

Tuesday, December 1	Friday, December 4
Wednesday, December 9	Friday, December 11
Tuesday, December 15	Thursday, December 17
Monday, December 21	Thursday, December 24

Four round-trip tickets must be purchased. Typical tickets cost \$250, but if the round-trip includes a weekend, the cost is discounted to \$200. Furthermore, if the round-trip includes at least 14 days in a city, the cost is reduced further to \$175. Formulate and solve a network model that minimizes the total ticket cost for these trips. *Hint:* Even though you work in Indianapolis, your round-trip tickets can begin with a flight from Boston.

2.36 Velvet Ale is produced by a local brewer. Currently, it has three production plants in town, one that can produce 1000 bottles a day, another that produces 750 bottles per day, while the third produces only 500 bottles per day. The brewer uses two distributors to deliver its beer to the three stores that sell it. The (per day) demand for the three stores is 700, 600, and 800, respectively. In addition, cost (in cents per bottle) to ship the beer between locations is given in the table below.

	Distributor 1	Distributor 2	Store 1	Store 2	Store 3
Plant 1	8	14	—	—	—
Plant 2	12	10	—	—	—
Plant 3	16	12	—	—	—
Distributor 1	—	—	10	8	12
Distributor 2	—	—	6	15	9

Formulate and solve a minimum cost network flow model to find the optimal routing of the beer from the brewer to the stores.

2.37 Continuing in Exercise 2.36, suppose that the distributors each want to stockpile bottles of beer in order to satisfy an additional (and unexpected) demand they may face. It costs distributor 1 \$0.50 per bottle in daily storage and distributor 2 \$0.35 per bottle, and each distributor can store at most 250 bottles. Formulate and solve a minimum cost network flow model to determine how Velvet Ale can minimize their costs.

2.38 Suppose that in Exercise 2.36 Velvet Ale has added a second beer to their production. To handle this, they have doubled each plant's production

capacity. The three stores have added a daily demand of 500, 900, and 700 bottles of this new product. Unfortunately, the trucks carrying the beer from the distributors to the stores can hold at most 1000 bottles each. Formulate and solve a linear program to find the new optimal routing of the beer from the brewer to the stores.

2.39 Suppose it costs \$20,000 to purchase a new car. The annual operating cost and resale value of a used car, based upon the age of a car, are given below. Assuming that you have a new car at present, determine a replacement policy that minimizes your net costs of owning and operating a car for the next 6 years.

Age of Car	Resale Value	Operating Cost
1	\$16,000	\$500
2	\$13,000	\$800
3	\$9000	\$1100
4	\$7000	\$1400
5	\$5500	\$1800
6	\$4000	\$2400

2.40 (Based on Frank [39]) BuildIt Construction uses steel beams with a uniform cross section but of different lengths in its various projects. For its projects over the next month it will require beams of lengths 4, 7, 9, 10, 14, 18, 23, and 30; the number of beams needed are 40, 30, 60, 110, 100, 75, 90, and 125, respectively. While it can hold all required beams in its storage center, BuildIt typically will store only some of these beam lengths and use them to satisfy the demand for smaller lengths, with the scrap pieces being discarded; for example, an 18 ft beam can be trimmed to satisfy all required lengths except for the 23 and 30 ft beams. It costs BuildIt \$100 to prepare the storage center for holding beams of any particular length, a per-beam amount of which is given below for each beam length.

	4 ft	7 ft	9 ft	10 ft	14 ft	18 ft	23 ft	30 ft
Cost/beam (\$)	8	9	11	13	20	25	32	38

Thus, if it were to hold 200 beams of length 18 ft, the cost would be $\$100 + 200(\$25) = \$5100$. If BuildIt wants to minimize the total cost of storing the required beams, formulate and solve this problem as a shortest path

problem. Hint: If we were to store beams of lengths 10 and 30 ft, would we use a beam of length 30 ft to satisfy the demand for a beam of length 7 ft in an optimal solution?

2.41 BlackGold Oil Company needs to ship oil each day through its pipelines in Hillbilly, MO to Beverly Hills, CA through any or all its monitoring stations. The network of its stations and pipelines is given in [Figure 2.5](#). Each arc in the network represents a pipeline with a corresponding maximum capacity (in thousands of gallons), which is due to the different pipeline diameters. Formulate and solve a linear program to determine the maximum amount of oil that can be shipped through this network from Hillbilly to Beverly Hills.

FIGURE 2.5 Network for BlackGold Oil in Exercise 2.41.

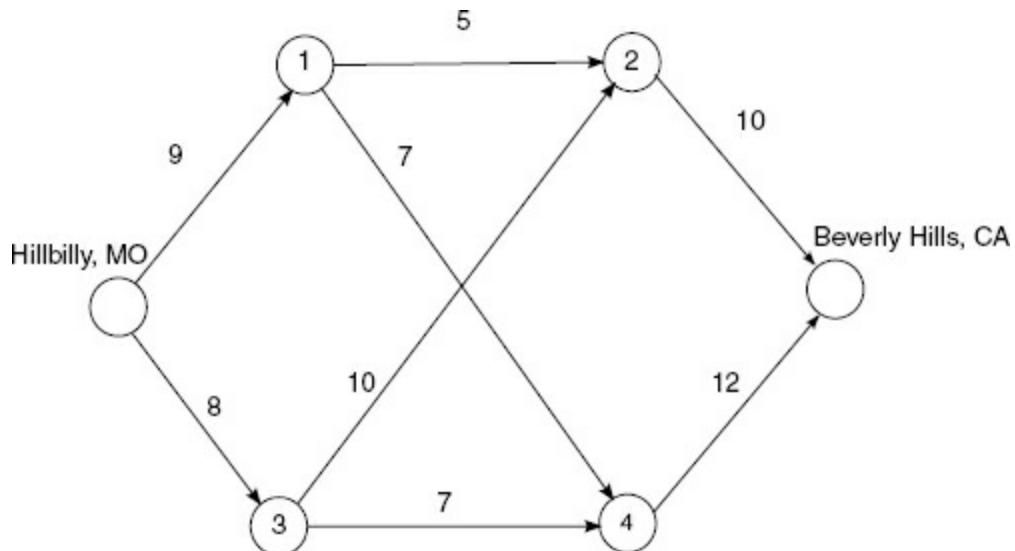
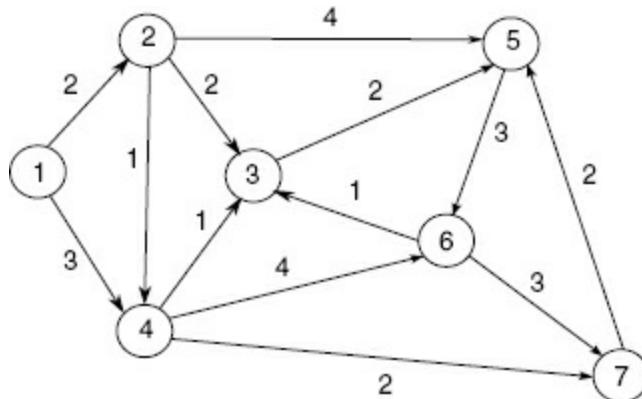


FIGURE 2.6 Shortest path problem for Exercise 2.42.



2.42 Consider the directed graph $G = (V, A)$ given in [Figure 2.6](#), where

each directed arc $(i, j) \in A$ has associated with it a distance d_{ij} . Formulate a minimum cost network flow model that will identify the shortest paths and their lengths from 1 to every other vertex in G .

2.43 Given a directed graph $G = (V, A)$ where each directed arc $(i, j) \in A$ has associated with it a distance d_{ij} . Assume that there exists a path P from vertex 1 to every other vertex j . Formulate a general minimum cost network flow model that will identify the shortest paths and their lengths from 1 to every other vertex in G .

2.44 Suppose you are given a transportation problem where total supply is more than total demand, that is,

$\sum_i s_i > \sum_j d_j$. Transform this problem into one that has equal total supply and demand by adding at most one new node. Now do the same if the total demand is greater than the total supply.

2.45 Suppose you are given a transportation problem where total supply is less than total demand, that is,

$\sum_i s_i < \sum_j d_j$. Note that the problem would have been infeasible. Transform this problem into a feasible problem that has equal total supply and demand by adding at most one new node. What is a solution to this problem indicating to us (in terms of the original model)?

2.46 Suppose we have a variable x that has a nonzero lower bound $x \geq a$. Transform this bound into an equivalent one where the variable has a lower bound of 0.

2.47 Given the feasible region $\{(x, y) : x - y = 0, x \geq 0, y \geq 0\}$ show that the bounds $x \geq 0$ and $y \geq 0$ are both redundant, but that both cannot be removed without altering the feasible region.

2.48 Given a maximization problem

$$\max \{f(\mathbf{x}) : \mathbf{x} \in S\},$$

for which $S = \emptyset$ and there exists a finite optimal solution \mathbf{x}^* , derive a minimization problem with S as its feasible region for which \mathbf{x}^* is an optimal solution.

2.49 In Section 2.7 we saw that one way to model $|x|$ was to use variables x^+ and x^- , add the constraint $x = x^+ - x^-$, and use the fact that $|x| = x^+ + x^-$.

Use the fact that $|x|$ is the smallest value z satisfying both $x \leq z$ and $-x \leq z$ to derive an additional formulation for $|x|$. Use this new approach to linearize(2.8).

CHAPTER 3

INTEGER AND COMBINATORIAL MODELS

As we saw in Chapter 2, many situations require the decision variables to have integer value. While the integer variables played no formal role in these models (they were just additional constraints), there are numerous occasions where integer variables are very integral to the model. Typically, though not always, these integer variables will have values either 0 or 1; such variables are called **binary variables**. These variables can handle such questions as “Do I make this product or not?,” “Should we locate our plant here?,” and so on. In this chapter, we examine different ways in which binary variables are used in the modeling process.

3.1 FIXED-CHARGE MODELS

Often in production and location problems, a one-time charge is applied when an activity is performed at some nonzero level. This charge does not depend on the amount of the activity, only that the activity is performed. For example, we may be interested in whether to operate a machine that requires a certain amount of setup time before it can produce any product, or when we must decide which plants and warehouses to open in order to meet all customer demands. Such problems come under the heading of *Fixed-Charge Models*. In this section, we will look at two different fixed-charge models, one from warehouse location and the other from production.

Warehouse Location Problems A common model that uses fixed charges is transshipment problems, where decisions must be made concerning which plants and warehouses to use in order to meet the given demands; such problems are often referred to as *Warehouse Location Problems*. Below is an

example of such a problem.

■ EXAMPLE 3.1

Recall the model given in Example 2.12, where PedalMetal Motors was shipping cars from two plants to three cities. Suppose that a new plant in San Antonio (SA) has been opened recently, with a capacity of 120 cars per week, as well as a new warehouse in New Orleans (NO). In addition, Detroit has increased its capacity to 150 cars per week. Costs to ship from the plants (including San Antonio) to the warehouses and from New Orleans to all customers are given below.

	DEN	CIN	NO	LA	CHI	PHI
DET	\$1253	\$637	\$1128	—	—	—
ATL	\$1398	\$841	\$702	—	—	—
SA	\$942	\$1154	\$691	—	—	—
NO	—	—	—	\$1983	\$1272	\$1751

Management now has options for where to produce and hold the cars before meeting customer demands. Therefore, they have decided that, for this make of car, PedalMetal Motors will not necessarily produce cars at all three plants nor use all three warehouses. In order to prepare each plant and warehouse for the new car, an initial (one-time) setup cost is incurred. These amounts are given below.

	DET	ATL	SA	DEN	CIN
Setup cost	\$20,000	\$18,000	\$10,000	\$5000	\$6000

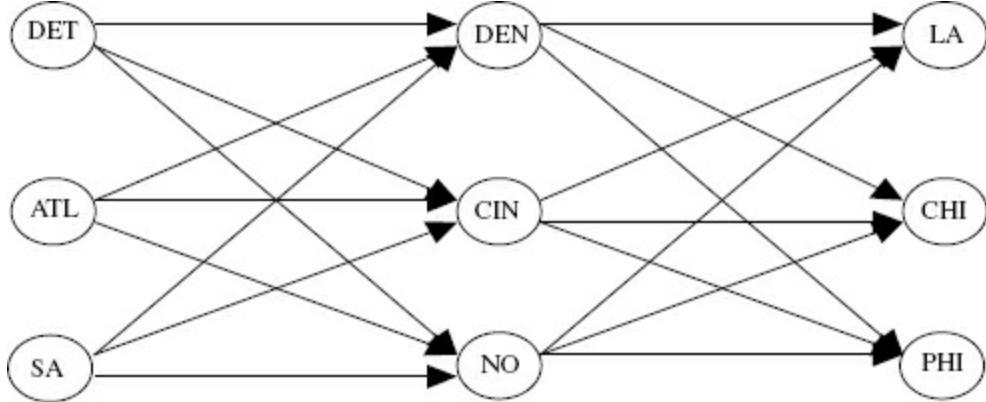
Which plants and warehouses should be opened in order to meet customer demands at minimum cost?

We will use a similar network as given in Chapter 2, with the addition of nodes for San Antonio and New Orleans. This network is shown in [Figure 3.1](#).

Again, without the setup costs, this would be a standard transshipment problem. So, how do we handle the setup costs? Hopefully, you can see that this is a prime example for the use of binary variables, where a variable takes a value of 1 if a plant or warehouse is to open and 0 if it is to remain closed. To this end, let's define our variables as

$$y_i = \begin{cases} 1, & \text{if plant/warehouse } i \text{ is open,} \\ 0, & \text{otherwise,} \end{cases}$$

FIGURE 3.1 Network for warehouse location problem.



where $i \in \{DET, ATL, SA, DEN, CIN, NO\}$. Using these variables, our setup costs are

$$20,000y_{DET} + 18,000y_{ATL} + 10,000y_{SA} + 5000y_{DEN} + 6000y_{CIN} + 5000y_{NO}.$$

How do we use these variables to limit shipments from unopened plants and warehouses? We'll first examine the plants and use Detroit as an example. Before, we had the constraint

$$x_{DET,DEN} + x_{DET,CIN} + x_{DET,NO} \leq 150,$$

but now we want the restrictions

$$x_{DET,DEN} + x_{DET,CIN} + x_{DET,NO} \begin{cases} = 0, & \text{if Detroit not open } (y_{DET} = 0), \\ \leq 150, & \text{if Detroit open } (y_{DET} = 1). \end{cases}$$

We can formulate this as

$$x_{DET,DEN} + x_{DET,CIN} + x_{DET,NO} \leq 150y_{DET}.$$

This works since each $x_{ij} \geq 0$, and the sum of multiple nonnegative variables equals 0 if and only if each variable has value 0; thus, if $y_{DET} = 0$, we must have that each of the variables on the left-hand side of the constraint has value 0. In other words, we ship nothing out of plant 1 if it is not open. We would have similar constraints for the other plants.

How about the warehouses? If a warehouse is not open, we cannot ship anything to them. Thus, using warehouse 3 (Denver), we'd want

$$x_{DET,DEN} + x_{ATL,DEN} + x_{SA,DEN} \begin{cases} = 0, & \text{if } y_{DEN} = 0, \\ \text{unrestricted}, & \text{if } y_{DEN} = 1. \end{cases}$$

How do we deal with the unrestricted value if Denver is open? The most that can be shipped to Denver is the sum of the supplies of each plant. Since the total that can be produced and shipped is at most 370 cars (150 + 100 + 120), we'd have the constraint

$$x_{DET,DEN} + x_{ATL,DEN} + x_{SA,DEN} \leq 370y_{DEN}.$$

In fact, we could use any coefficient for y_{DEN} as long as it is not smaller than 370. Of course, we still need to have the balance constraints

$$x_{DET,DEN} + x_{ATL,DEN} + x_{SA,DEN} = x_{DEN,LA} + x_{DEN,CHI} + x_{DEN,PHI},$$

to ensure the node remains a transshipment node; there would be similar constraints for each warehouse. Our integer program is given in Integer Program 3.1.

Integer Program 3.1 Integer Program Model for Example 3.1

$$\begin{aligned}
\min \quad & 10000(x_{DET,DEN} + x_{DET,CIN} + x_{DET,NO} + x_{ATL,DEN} + x_{ATL,CIN}) \\
& + 10000(x_{ATL,NO} + x_{SA,DEN} + x_{SA,CIN} + x_{SA,NO}) \\
& + 1253x_{DET,DEN} + 637x_{DET,CIN} + \dots + 1983x_{NO,LA} \\
& + 1272x_{NO,CHI} + 1751x_{NO,PHI} \\
& + 20000y_{DET} + 18000y_{ATL} + 10000y_{SA} \\
& + 5000y_{DEN} + 6000y_{CIN} + 5000y_{NO} \\
\text{s.t.} \quad & x_{DET,DEN} + x_{DET,CIN} + x_{DET,NO} \leq 150y_{DET} \\
& x_{ATL,DEN} + x_{ATL,CIN} + x_{ATL,NO} \leq 100y_{ATL} \\
& x_{SA,DEN} + x_{SA,CIN} + x_{SA,NO} \leq 120y_{SA} \\
& x_{DET,DEN} + x_{ATL,DEN} + x_{SA,DEN} = x_{DEN,LA} + x_{DEN,CHI} + x_{DEN,PHI} \\
& x_{DET,DEN} + x_{ATL,DEN} + x_{SA,DEN} \leq 370y_{DEN} \\
& x_{DET,CIN} + x_{ATL,CIN} + x_{SA,CIN} = x_{CIN,LA} + x_{CIN,CHI} + x_{CIN,PHI} \\
& x_{DET,CIN} + x_{ATL,CIN} + x_{SA,CIN} \leq 370y_{CIN} \\
& x_{DET,NO} + x_{ATL,NO} + x_{SA,NO} = x_{NO,LA} + x_{NO,CHI} + x_{NO,PHI} \\
& x_{DET,NO} + x_{ATL,NO} + x_{SA,NO} \leq 370y_{NO} \\
& x_{DEN,LA} + x_{CIN,LA} + x_{NO,LA} \geq 80 \\
& x_{DEN,CHI} + x_{CIN,CHI} + x_{NO,CHI} \geq 70 \\
& x_{DEN,PHI} + x_{CIN,PHI} + x_{NO,PHI} \geq 60 \\
& x_{ij} \geq 0 \\
& y_j \in \{0, 1\}
\end{aligned}$$

Solution An optimal solution indicates that the plants in Detroit and San Antonio should be open, as well as the warehouses in Cincinnati and Denver. We should send 130 cars from Detroit to Cincinnati and 80 from San Antonio and Denver, and then 70 cars from Cincinnati to Chicago, 60 from Cincinnati to Philadelphia, and 80 from Denver to Los Angeles. The optimal cost for this shipping schedule is \$2,482,030.

Production Fixed-Charge Problem In production fixed-charge problems, we are often interested in determining which machines to operate so as to meet given requirements and either minimize costs or maximize profits.

■ EXAMPLE 3.2

The bakery inside the local grocery needs to determine which of its three ovens to operate in order to produce its fresh rolls. For each oven, there is a one-time start-up cost associated with preparing the oven and cleaning it afterward. In addition, there is a cost per roll produced, which differs by oven. There are two baking periods in a day, and the number of rolls needed in the first period is 500, while the second period requires 600. An oven started in the first period can be used in the second period without incurring an additional start-up cost. Below are the characteristics of the ovens.

Oven	Start-Up Cost (\$)	Cost Per Period Per Roll	Maximum Capacity Per Period
1	350	0.08	360
2	400	0.04	360
3	250	0.12	280

How should the bakery proceed to meet demand at minimum cost?

In this problem, we have two layers of decisions. First, we must decide which ovens to operate in each period, and then we decide how many rolls to generate from each operating oven. Let's first tackle the issue of which ovens to operate, using an approach similar to that from Example 3.1. All variables we define will take value 1 if we use the oven and 0 otherwise. Also, we need to be careful, since it is entirely possible for an oven to be used in period 1 and not period 2. Although this seems unlikely, we must account for this possibility. Since we need indicator variables for each oven and each period, let's define our variables as

$$x_{ij} = \begin{cases} 1, & \text{if oven } i \text{ is first used in period } j, \\ 0, & \text{otherwise,} \end{cases}$$

$$y_i = \begin{cases} 1, & \text{if oven } i \text{ is used in period 2,} \\ 0, & \text{otherwise.} \end{cases}$$

Why do the variable definitions x_{ij} include the word “first”? Since each oven can be used in both periods, and we only incur at setup cost the first time it is used, we need to identify when the first time is. Using oven 1 as an example, note that, since we can use only a generator “first” once, we have

$$x_{11} + x_{12} \leq 1.$$

Of course, there would be similar constraints for ovens 2 and 3. Also, we cannot use the oven in period 2 if it has not been first used in either period 1 or 2. This yields the constraint

$$(3.1) \quad y_1 \leq x_{11} + x_{12},$$

which indicates that y_1 cannot take value 1 unless one of the variables x_{11} , x_{12} has value 1.

Now that we know how to recognize which ovens are used in each period, let's consider how many rolls each oven can produce. This leads to the variables

$$R_{ij} = \text{number of rolls produced by oven } i \text{ in period } j,$$

for ovens $i = 1, 2, 3$ and periods $j = 1, 2$. How do we limit production from the ovens not used in a given period? Without setup costs, we'd have the constraints

$$R_{11} \leq 360, \quad R_{12} \leq 360$$

$$R_{21} \leq 360, \quad R_{22} \leq 360$$

$$R_{31} \leq 280, \quad R_{32} \leq 280.$$

However, this does not limit production when an oven is not operating. To remedy this, notice that we want, using oven 1 as an example,

$$(3.2) \quad R_{11} = \begin{cases} = 0, & \text{if } x_{11} = 0, \\ \leq 360, & \text{if } x_{11} = 1. \end{cases}$$

We can formulate this as the constraint

$$R_{11} \leq 360x_{11}.$$

Using the fact that $R_{11} \geq 0$, we guarantee that (3.2) holds. For period 2, we'd have the similar constraint

$$R_{12} \leq 360y_1.$$

Now to ensure we meet the demand for each period, we have the constraints

$$R_{11} + R_{21} + R_{31} \geq 500,$$

$$R_{12} + R_{22} + R_{32} \geq 600.$$

Our objective is to minimize costs, which include both setup and production costs. Setup costs are calculated as

$$350(x_{11} + x_{12}) + 400(x_{21} + x_{22}) + 250(x_{31} + x_{32}),$$

and production costs are

$$0.08(R_{11} + R_{12}) + 0.04(R_{21} + R_{22}) + 0.12(R_{31} + R_{32}).$$

Thus, our (binary) integer program is given in Integer Program 3.2.

Integer Program 3.2 Integer Program for Example 3.2

$$\min \quad 0.08(R_{11} + R_{12}) + 0.04(R_{21} + R_{22}) + 0.12(R_{31} + R_{32})$$

$$+ 350(x_{11} + x_{12}) + 400(x_{21} + x_{22}) + 250(x_{31} + x_{32})$$

s.t.

$$\sum_{j=1}^2 x_{ij} \leq 1, \quad i \in \{1, 2, 3\}$$

$$y_i \leq \sum_{j=1}^2 x_{ij}, \quad i \in \{1, 2, 3\}$$

$$R_{11} \leq 360x_{11}$$

$$R_{12} \leq 360y_1$$

$$R_{21} \leq 360x_{21}$$

$$R_{22} \leq 360y_2$$

$$R_{31} \leq 280x_{31}$$

$$R_{32} \leq 280y_3$$

$$R_{11} + R_{21} + R_{31} \geq 500$$

$$R_{12} + R_{22} + R_{32} \geq 600$$

$$x_{A1}, x_{A2}, x_{B1}, x_{B2}, x_{C1}, x_{C2} \geq 0$$

$$x_{ij}, y_i, R_{ij} \in \{0, 1\}, \quad i \in \{1, 2, 3\}, j \in \{1, 2\}$$

Solution The optimal solution indicates that ovens 1 and 3 are used in both periods, while oven 2 is not used in any period. In period 1, 360 rolls are produced by oven 1 and 140 rolls are produced by oven 3. In period 2, 360 rolls are produced by oven 1 and 240 rolls are produced by oven 3. This yields an optimal cost of \$703.20. Note that, even with its low per-roll cost, oven 2 is not used in this solution.

General Model In each of the two examples above, we saw that fixed-charge models use binary variables to restrict values of other variables. Let's

consider the linear function $\sum_j a_j x_j$, where each coefficient $a_j > 0$ and each variable $x_j \geq 0$. We are interested in modeling whether it has positive value. If we define $y \in \{0, 1\}$ to be

$$y = \begin{cases} 1, & \text{if } \sum_j a_j x_j > 0, \\ 0, & \text{otherwise,} \end{cases}$$

we can model this situation as

$$(3.3) \quad \sum_j a_j x_j \leq M y$$

for some large number M ; such a number is referred to as “Big- M .” As discussed before, since each a_j and x_j is nonnegative, if $y = 0$, then we must have each $x_j = 0$. If $y = 1$, we allow any (and/or all) x_j to be positive (again, provided M is “large enough”). We can further restrict the values of each x_j by putting an upper bound on the function, similar to what we did in the examples.

These fixed-charge models are really just explicit examples of a general class of *logical constraints*. These constraints relate binary variables to one another. We will explore more of these in Section 3.3.

3.2 SET COVERING MODELS

Selecting a few from a collection of many is a common situation. For example, if we are forming a subcommittee to determine recent financial decisions on employee morale, it seems reasonable to choose at least one employee from each department. Such problems are known as *Set Covering Problems*, where the set is the collection of choices from which at least one must be chosen.

■ EXAMPLE 3.3

A local community is looking to put precinct police stations throughout the city. The administration has divided the city into 10 districts and has designated a list of six candidate locations for the stations. Each station can

serve a small number of the districts, as described in the table below.

Station	Districts Served
1	1, 3, 5, 6
2	1, 4, 7, 9
3	2, 5, 8
4	2, 3, 4, 7
5	4 6 10

The administration wants to determine the fewest number of stations to open while still serving all 10 districts. Which ones should they open?

Again, binary variables are important to this class of models. Here, we have a variable x_i defined for each station i with value definitions

$$x_i = \begin{cases} 1, & \text{if station } i \text{ is selected,} \\ 0, & \text{otherwise.} \end{cases}$$

Once we have these variables, the constraints seem straightforward. For example, to service district 1, either station 1 or station 2 must be open, giving the constraint

$$x_1 + x_2 \geq 1.$$

If we look at district 4, it can be serviced by station 2, 4, or 5, giving the constraint

$$x_2 + x_4 + x_5 \geq 1.$$

Each of these constraints indicate that at least one of the variables must have value 1. Such constraints are known as **Covering Constraints**.

Our objective function is to minimize the sum of the variables, that is, to minimize the number of stations opened. This gives the set covering integer program found in Integer Program 3.3.

Integer Program 3.3 Integer Program for Example 3.3

$$\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \geq 1 \quad (\text{District 1}) \\
& x_3 + x_4 \geq 1 \quad (\text{District 2}) \\
& x_1 + x_4 \geq 1 \quad (\text{District 3}) \\
& x_2 + x_4 + x_5 \geq 1 \quad (\text{District 4}) \\
& x_1 + x_3 \geq 1 \quad (\text{District 5}) \\
& x_1 + x_5 \geq 1 \quad (\text{District 6}) \\
& x_2 + x_4 \geq 1 \quad (\text{District 7}) \\
& x_3 + x_6 \geq 1 \quad (\text{District 8}) \\
& x_2 + x_6 \geq 1 \quad (\text{District 9}) \\
& x_5 + x_6 \geq 1 \quad (\text{District 10}) \\
& x_i \in \{0, 1\}, \quad i \in \{1, 2, \dots, 6\}
\end{aligned}$$

Solution The minimum number of stations to open is 3, where each district is covered by at least one of the stations 1, 4, or 6.

General Model When we model with binary variables, we often have situations like the one mentioned in this section where we need to ensure that at least one variable from a collection has value 1. In fact, you can probably also see that cases where we choose either at most one or exactly one variable to have value 1 also frequently arise in our models. Each of these cases appear so often that they have specific names associated with them: (1) covering constraints, (2) packing constraints, and (3) partitioning constraints.

Given a collection C of choices, where we define variable

$$x_k = \begin{cases} 1, & \text{if choice } k \text{ is selected,} \\ 0, & \text{otherwise,} \end{cases}$$

a **covering constraint** is one of the form

$$\sum_{i \in C} x_i \geq 1;$$

a **packing constraint** is one of the form

$$\sum_{i \in C} x_i \leq 1;$$

and a **partitioning constraint** is one of the form

$$\sum_{i \in C} x_i = 1.$$

3.3 MODELS USING LOGICAL CONSTRAINTS

Some models naturally have constraints for which at least one must be satisfied, but not necessarily both. Such constraints are often known as *Either–Or Constraints*. It should seem reasonable that binary variables can be used to determine which constraint must be satisfied. Our example in this section illustrates this point.

■ EXAMPLE 3.4

Custom Benches produces and finishes various wooden outdoor furniture. After each piece has been produced, it goes to its Finishing Department, where various workstations apply various paints and stains to produce the final product. Each piece requires time at various combinations of four different workstations, with each job visiting the workstations in different orders. There are currently two types of furniture needing to be scheduled: (1) Adirondack Chairs and (2) Bench Swings. Below are the sequence of workstations and processing times (in minutes) required for each job. Each workstation can work only on one job at a time and must complete each job before beginning a new one.

Adirondack Chairs		Bench Swings	
Workstation	Time	Workstation	Time
1	3	2	8
2	10	3	6
3	11	1	8

If the company wants to complete both jobs as soon as possible, how should it schedule them?

The above example deals with a **job shop scheduling problem**, which seeks

an optimal solution for a collection of jobs over multiple machines. Each job goes through a known sequence of machines, and each machine can handle only one job at a time. For such problems, there are various objective functions, but the most common is probably to minimize the *makespan*, in other words, to determine the starting times for each job on each machine in order to minimize the maximum completion time of all jobs.

In our problem, we have two jobs and four different machines or workstations. We assume that “Adirondack Chairs” is job 1 and “Bench Swings” is job 2. Since we need to identify the start times for each job $j \in \{1, 2\}$ at each workstation $k \in \{1, 2, 3, 4\}$, we define our variables as

$$x_{jk} = \text{start time of job } j \text{ on workstation } k.$$

The amount of time it takes to finish job 1 is equal to the start time on workstation 4 plus the time required on workstation 4, that is, $x_{14} + 7$. Similarly, for job 2, the total time is $x_{24} + 5$. Thus, the total amount of time to complete both the jobs would be

$$\max\{x_{14} + 7, x_{24} + 5\}.$$

For constraints, we must be sure that the start times for each job are in the correct sequence, because we cannot start job 1 on workstation 2 before it is completed on workstation 1. For job 1, we would have the following **precedence constraints**:

$$x_{11} + 3 \leq x_{12}$$

$$x_{12} + 10 \leq x_{13}$$

$$x_{13} + 11 \leq x_{14}$$

Similarly, for job 2, working with the workstation order 2–3–1–4, we’d have the precedence constraints

$$x_{22} + 8 \leq x_{23}$$

$$x_{23} + 6 \leq x_{21}$$

$$x_{21} + 8 \leq x_{24}$$

These constraints are not equality constraints because we do not know if we’ll be able to immediately place a job on the next workstation, since the workstation may be working on the other job at that time.

More constraints are needed, since we have not placed any restrictions on the number of jobs a workstation can work on at any given time. For this, we

identify, for each workstation, which job is scheduled first by using the following binary variables:

$$y_{j,j',k} = \begin{cases} 1, & \text{if job } j \text{ is scheduled before job } j' \text{ on work } k \\ 0, & \text{otherwise,} \end{cases}$$

where $j < j'$. How do we use these? Let's take workstation 1 as an example. Either job 1 or job 2 is scheduled first, and the other job cannot be started until the first job is completed. If job 1 is scheduled first, then the following constraint must hold:

$$x_{11} + 3 \leq x_{21}.$$

If job 2 is scheduled first, we have

$$x_{21} + 8 \leq x_{11}.$$

At least one of these constraints must hold and, using the $y_{j,j',k}$ variables, we can model this as

$$\begin{aligned} x_{11} + 3 &\leq x_{21} + M(1 - y_{121}) \\ x_{21} + 8 &\leq x_{11} + My_{121}, \end{aligned}$$

where M is a “large enough” integer. This is an example of either-or constraints. Why does this technique work? If we want job 1 to be first, then setting $y_{121} = 1$ generates the constraints

$$\begin{aligned} x_{11} + 3 &\leq x_{21} \\ x_{21} + 8 &\leq x_{11} + M. \end{aligned}$$

We are left with the first constraint that must hold, while the second one always holds as long as M is large enough. If we want job 2 to go first, we'd set $y_{121} = 0$ and get the constraints

$$\begin{aligned} x_{11} + 3 &\leq x_{21} + M \\ x_{21} + 8 &\leq x_{11}. \end{aligned}$$

We would have these either-or constraints for each workstation where there could be a conflict in scheduling.

Recalling that our objective is a minimax function, our integer program is found in Integer Program 3.4.

Integer Program 3.4 Integer Program for Example 3.4

$$\begin{aligned}
\min \quad & \lambda \\
\text{s.t.} \quad & \\
& x_{11} + 3 \leq x_{12} \quad (\text{Job 1 precedence}) \\
& x_{12} + 10 \leq x_{13} \\
& x_{13} + 11 \leq x_{14} \\
& x_{22} + 8 \leq x_{23} \quad (\text{Job 2 precedence}) \\
& x_{23} + 6 \leq x_{21} \\
& x_{21} + 8 \leq x_{24} \\
& x_{11} + 3 \leq x_{21} + M(1 - y_{121}) \quad (\text{Workstation 1 conflicts}) \\
& x_{21} + 8 \leq x_{11} + My_{121} \\
& x_{12} + 10 \leq x_{22} + M(1 - y_{122}) \quad (\text{Workstation 2 conflicts}) \\
& x_{22} + 8 \leq x_{12} + My_{122} \\
& x_{13} + 11 \leq x_{23} + M(1 - y_{123}) \quad (\text{Workstation 3 conflicts}) \\
& x_{23} + 6 \leq x_{13} + My_{123} \\
& x_{14} + 7 \leq x_{24} + M(1 - y_{124}) \quad (\text{Workstation 4 conflicts}) \\
& x_{24} + 5 \leq x_{14} + My_{124} \\
& \lambda \geq x_{14} + 7 \quad (\text{Minimax constraints}) \\
& \lambda \geq x_{24} + 5 \\
& x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{24}, \lambda \geq 0 \\
& y_{121}, y_{122}, y_{124} \in \{0, 1\}
\end{aligned}$$

Solution If we choose M to be the sum of all job times ($M = 58$), we find that the quickest both jobs can be completed is in 36 time units, where Adirondack Chairs are scheduled first on workstation 1, then has to wait for workstation 2 to finish with the Bench Swings before heading to the remaining workstation it needs without delay. Bench Swings goes on each workstation without delay.

It is interesting to see the difference between a theoretical model and how it behaves computationally. We know that, theoretically, as long as $M \geq 58$, our model should generate the optimal solution of 32 time units. However, if we choose, for example, $M = 10^{16}$, we get a solution where both the Chairs and Benches are scheduled on workstation 2 at time $t = 3$. This is because every computational package uses the notion of tolerance to indicate equality in an inequality. For example, if the tolerance is set to $\epsilon = 10^{-8}$ (a typical value), and if we have a value of $x = 10^{-10}$, then x is treated as having value 0, since $10^{-10} < \epsilon$. If we have $y_{122} = 10^{-9}$, for instance, it would be treated the same

numerically as $y_{122} = 0$, but with $M = 10^{16}$, we would have $My_{122} = 10^7 \neq 0$, which causes a huge problem in the actual solution of the model. Because of these numerical difficulties, it is important to keep the “ M ” in big- M as small as possible.

General Model Logical conditions, such as the either–or condition we just explored, appear in many types of modeling situations and can often be modeled using binary variables. We have already seen some of these cases; for example, the fixed-charge models from Section 3.1 and the covering constraints in Section 3.2 are logical conditions. Let’s explore (and review) some of these logical conditions and how they can be modeled using binary variables.

Suppose we have the two constraints

$$\begin{aligned} f &\leq 0 \\ g &\leq 0, \end{aligned}$$

and assume that at least one of these must hold. Letting y be a binary variable, we can define the two constraints

$$\begin{aligned} f &\leq M(1 - y) \\ g &\leq My \end{aligned}$$

and make both constraints be satisfied. If $y = 1$, then the constraint $f \leq 0$ is forced to hold, while if $y = 0$, we make $g \leq 0$ hold.

Note that we can model the logical situation

$$y = 1 \Rightarrow f \leq 0$$

similar to our handling of the either–or constraints if we use only the constraint

$$f \leq M(1 - y).$$

How could we handle the logical condition

$$\sum_{j=1}^n a_j x_j \leq b \Rightarrow y = 1,$$

where each a_j and b are integers? Note that this statement is logically equivalent to

$$y = 0 \Rightarrow \sum_{j=1}^n a_j x_j \geq b + 1.$$

We can use the either-or conditions again to write the constraint

$$(b + 1) - \sum_{j=1}^n a_j x_j \leq M y.$$

We shall explore other logical conditions in Exercises 3.36 and 3.37. For more information on the modeling of logical conditions, a good reference to explore is Williams [85].

3.4 COMBINATORIAL MODELS

A different type of integer model comes from combinatorial problems that are easy to formulate combinatorially, but are difficult to formulate in the standard optimization model. Often, these problems are formulated on a graph or some other combinatorial object, and can easily be described in that context. In this section, we will examine two such problems. However, unlike in the previous sections, it is easier to initially formulate the problem in a general form than in a specific instance. Hence, we will be constructing both a specific model and the general form simultaneously; this will also give us practice in formulating the general model directly.

Minimum Spanning Trees

In many situations, we are interested in the minimum connectedness between groups or entities. We want to enable each entity to “communicate” with the others, but not have any redundancy in the connections.

Mathematically, such a problem is easily described using graph theory. We shall let the vertices V of a graph G denote those items that we wish to connect, and the edges E denote those items that can be directly linked. Recall that a graph G is connected if, for every pair of vertices $i, j \in V$, there is a path in G from i to j using only edges in E . If we have a connected graph G that contains a cycle, then the removal of any one edge from this cycle does not destroy the connectivity of the graph. Thus, a minimally connected graph G contains no cycles and is called a *tree*. A subgraph G' of G that is a tree and where every vertex in G is a vertex of G' is called a *spanning tree*. If

we associate a cost c_{ij} with each edge $(i, j) \in E$, then the problem of finding the spanning tree $G' = (V, T)$, where $T \subseteq E$, whose total cost

$$\sum_{(i,j) \in T} c_{ij}$$

is minimized is called the *Minimum Cost Spanning Tree Problem*. Such a problem is easy to formulate and arises in many connectivity problems.

■ EXAMPLE 3.5

A local phone company is interested in laying cable from the main road (where the main switch is located) to a new housing subdivision, and wants to do so in the least expensive way. It has the option of laying cable from the road to any house, or it can lay cable between the houses. Each house must be connected through some path to the road. The following matrix gives the total cost of laying cable between any two locations, where the first location is the main road.

$$C = \begin{bmatrix} 0 & 25 & 25 & 15 & 10 & 30 \\ 25 & 0 & 10 & 25 & 20 & 15 \\ 25 & 10 & 0 & 20 & 30 & 15 \\ 15 & 25 & 20 & 0 & 15 & 20 \\ 10 & 20 & 30 & 15 & 0 & 20 \\ 30 & 15 & 15 & 20 & 20 & 0 \end{bmatrix}.$$

How should the phone company connect the houses to the road in order to minimize its total cost?

While it is easy (and natural) to formulate this problem as a minimum cost spanning tree problem on a graph whose vertices are the homes and the road, let's instead construct an integer program that models this situation. To do so, let's define the variables

$$x_{ij} = \begin{cases} 1, & \text{if edge } (i, j) \text{ is in the spanning tree,} \\ 0, & \text{otherwise,} \end{cases}$$

for each $(i, j) \in E$. Note that every vertex i must be adjacent to at least one edge in the tree, giving the constraint

$$\sum_{j:(i,j) \in E} x_{ij} \geq 1.$$

In addition, since every spanning tree contains exactly $n - 1$ edges, where n

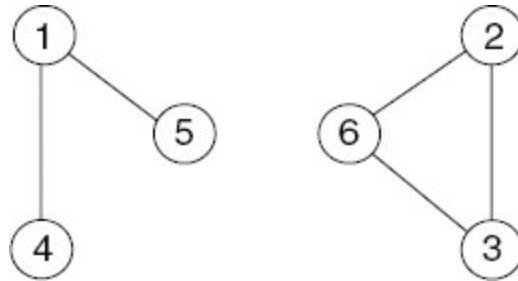
is the number of vertices, we have the constraint

$$\sum_{(i,j) \in E} x_{ij} = n - 1.$$

Our objective is to minimize the total cost $\sum_{(i,j) \in E} c_{ij}x_{ij}$, resulting in the integer program

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j) \in E} x_{ij} \geq 1, \quad i \in V \\ & \sum_{(i,j) \in E} x_{ij} = n - 1 \\ & x_{ij} \in \{0, 1\}. \end{aligned}$$

FIGURE 3.2 Initial solution to minimum spanning tree problem.



■ EXAMPLE 3.6

Consider the data from Example 3.5. Placing these data into the integer program (3.4) and solving yields the solution $x_{14} = x_{15} = x_{23} = x_{26} = x_{36} = 1$, with an objective value of 65. Note that this does not generate a spanning tree, since the resulting subgraph with edges $E' = \{(1, 4), (1, 5), (2, 3), (2, 6), (3, 6)\}$ is not connected and it contains the cycle $2 \rightarrow 3 \rightarrow 6 \rightarrow 2$ (see [Figure 3.2](#)).

What went wrong? We forgot to mention that, no matter what subset of vertices V' of V , we cannot have a cycle among V' as well. How can we correct this? One way to do this is to note that if there is a cycle among the vertices of V' , and there are $|V'| = n'$ vertices in V' , then there must be n' edges in the cycle. Thus, if we do not want to have a cycle among the vertices of V' , a necessary condition is that

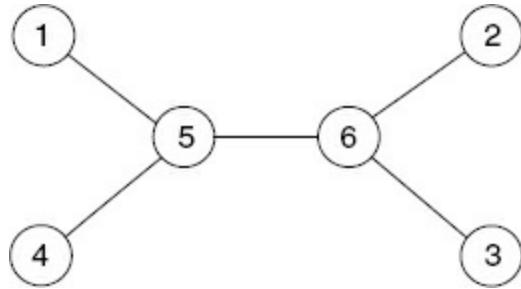
$$(3.4) \quad \sum_{\substack{i,j \in V' \\ (i,j) \in E}} x_{ij} \leq |V'| - 1.$$

If we add such a constraint for all subsets $V' \subset V$, this would eliminate the possibility of a feasible integer solution containing a cycle. The complete formulation of the problem is given in Integer Program 3.5.

Integer Program 3.5 IP Formulation of Minimum Spanning Tree Problem

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j) \in E} x_{ij} \geq 1, \quad i \in V \\ & \sum_{(i,j) \in E} x_{ij} = n - 1 \\ & \sum_{\substack{i,j \in V' \\ (i,j) \in E}} x_{ij} \leq |V'| - 1, \quad V' \subset V \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

FIGURE 3.3 Optimal solution to Example 3.5.



Of course, the number of constraints (3.4) can be prohibitive—if, for example, there are $n = 30$ nodes (a small problem), the number of such constraints approaches 2^{30} , which is approximately one billion. One option is to iteratively add such “cycle” constraints as needed; if we find a solution that represents a spanning tree using only some of the constraints (3.4) in our model formulation, we know that all other (unwritten) constraints are also satisfied.

■ EXAMPLE 3.7

Continuing from Example 3.6, if we add the constraint

$$x_{23} + x_{26} + x_{36} \leq 2$$

to our model and resolve, we get the solution $x_{14} = x_{15} = x_{45} = x_{23} = x_{36} = 1$, with a value of 65; note that this solution generates a cycle $1 \rightarrow 4 \rightarrow 5 \rightarrow 1$. If we add the cycle constraint

$$x_{14} + x_{15} + x_{45} \leq 2$$

and resolve, we get the solution $x_{15} = x_{23} = x_{36} = x_{45} = x_{56} = 1$, with a value of 70. This is a spanning tree, and hence our optimal solution. We did not need to add any other constraints (3.4) to find this solution. The minimal spanning tree is given in [Figure 3.3](#).

The minimum spanning tree problem is a classical combinatorial optimization problem, and has been well studied since the 1950s. Like the network flow models we saw in Section 2.9, combinatorial algorithms for the minimum spanning tree problem have been developed that are much more efficient computationally than any integer program formulation. We will explore such algorithms in Chapter 5. One of the reasons we mention the integer programming formulation of this problem is that if additional restrictions are placed on the problem (such as an upper bound on the number of edges incident to any node), it is typically solved using this formulation with additional constraints.

Traveling Salesperson Problems

Another classic combinatorial problem arises in the routing of a path or cycle that visits every vertex exactly once. Such a problem is found in many diverse applications such as airline scheduling, drilling printed circuit boards, pizza delivery systems, and so on. To illustrate this problem, consider the following example.

■ EXAMPLE 3.8

A college student is interested in visiting as many graduate schools as possible. She reasons that a single visit to each school is appropriate, and she wants to return to her own campus only after visiting all the schools. It is conceivable that she visits the schools in any order, but she would like to minimize the amount of driving she has to do. If the distance between schools i and j is d_{ij} , where the matrix D of distances is given below, in which order

should she visit the schools? Note that school 1 is her current school.

$$D = \begin{bmatrix} - & 16 & 23 & 14 & 8 & 15 \\ 16 & - & 12 & 19 & 9 & 13 \\ 23 & 12 & - & 7 & 25 & 16 \\ 14 & 19 & 7 & - & 18 & 15 \\ 8 & 9 & 25 & 18 & - & 20 \\ 15 & 13 & 16 & 15 & 20 & - \end{bmatrix}.$$

This **routing problem** is very common and well studied.

TSP Given a network $G = (V, A)$ and a cost matrix D , the *Traveling Salesperson Problem (TSP)* seeks a minimum cost route visiting each location exactly once.

In the TSP, any route visiting each location exactly once is known as a **tour**. The cost of the tour can also be viewed as the length of the tour, depending upon the application.

To formulate the TSP as an integer program, we need to be careful about which version we're interested in modeling. In Example 3.8, note that the matrix D , which gives the objective coefficients, is symmetric, in that, for all $i < j$, we have $d_{ij} = d_{ji}$. Such TSPs are called *Symmetric TSPs*. When they are not symmetric, the problem is the *Asymmetric TSP*. For now, we'll concentrate only on the symmetric case (see Exercise 3.32 for the asymmetric case).

There are many different formulations of the symmetric TSP, and none of them are straightforward. Most of them use as decision variables ($i < j$)

$$x_{ij} = \begin{cases} 1, & \text{if the route includes a leg between } i \text{ and } j, \\ 0, & \text{otherwise.} \end{cases}$$

Since everything is symmetric, we can disregard variables x_{ij} where $j < i$. Thus, the total cost/length of a tour is

$$\sum_i \sum_{j>i} d_{ij} x_{ij}.$$

In addition, note that, for the symmetric case, the order of the trip is not important. For example, a trip from cities $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ has the same cost/length as the trip $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

Similar to the minimum spanning tree problem, what makes TSP formulations so interesting (and frustratingly difficult!) is the constraints.

Some of the constraints are simple, while others are more complex. For example, it may appear obvious that, since we are looking for a tour of the cities, exactly two legs of the tour must be adjacent to every city. Mathematically, this would give the constraint

$$(3.5) \quad \sum_{i < j} x_{ij} + \sum_{i > j} x_{ji} = 2, \quad i \in V.$$

Are these enough constraints to completely describe the TSP? No. (Oh, but wouldn't that be nice!)

■ EXAMPLE 3.9

Consider the graph on six nodes in [Figure 3.4](#), where the numbers on each edge correspond to the cost of traversing that edge. If we used only constraints of the form (3.5), our optimization model would be

$$\begin{aligned} \text{min} \quad & 10x_{12} + x_{13} + x_{15} + x_{24} + x_{26} + 10x_{34} + x_{35} + x_{46} + 10x_{56} \\ \text{s.t.} \quad & \end{aligned}$$

$$x_{12} + x_{13} + x_{15} = 2$$

$$x_{12} + x_{24} + x_{26} = 2$$

$$x_{13} + x_{34} + x_{35} = 2$$

$$x_{24} + x_{34} + x_{46} = 2$$

$$x_{15} + x_{35} + x_{56} = 2$$

$$x_{26} + x_{46} + x_{56} = 2$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in E.$$

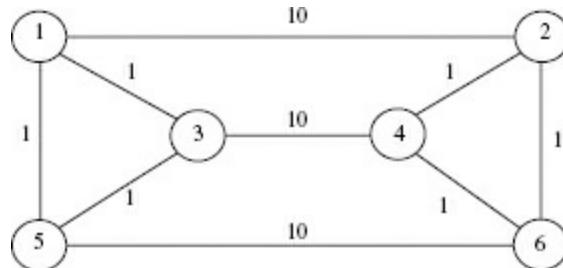
You should be able to verify logically that the following feasible solution is optimal:

$$x_{13} = x_{35} = x_{15} = 1$$

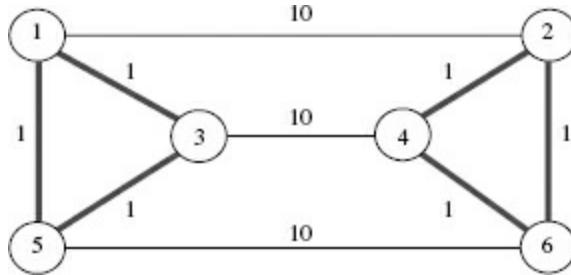
$$x_{24} = x_{46} = x_{26} = 1$$

$$x_{12} = x_{34} = x_{56} = 0.$$

[FIGURE 3.4](#) Symmetric TSP in Example 3.9.



[FIGURE 3.5](#) Solution to symmetric TSP with subtours.



This solution is given graphically in [Figure 3.5](#), where the “tour” is shown as thick edges.

What Example 3.9 demonstrated is that constraints (3.5) are not enough to model TSPs. In fact, our solution identified two **subtours** or miniroutes that may arise from these constraints. Hence, we need constraints that eliminate these subtours.

How do we do this? Consider the subtour $1 \rightarrow 3 \rightarrow 5$. There are three different legs given here, but in a “real tour,” there can only be at most two legs among the three possible ones here; otherwise, we’d have a subtour. The same holds true for the subtour $2 \rightarrow 4 \rightarrow 6$. This implies constraints of the form

$$(3.6) \quad x_{13} + x_{15} + x_{35} \leq 2$$

$$(3.7) \quad x_{24} + x_{26} + x_{46} \leq 2.$$

Can we generalize this idea? Suppose V denotes all cities in the problem and let $S \subset V$, that is, a proper subset of V . Among these cities, at most $|S| - 1$ legs can be chosen, otherwise we could have a subtour. This leads to the following set of constraints:

$$(3.8) \quad \sum_{\substack{i,j \in S \\ (i,j) \in E}} x_{ij} \leq |S| - 1, \quad \text{for all } S \subset V, \quad 3 \leq |S| \leq |V| - 3.$$

Constraints of the form (3.8) are commonly referred to as **subtour elimination constraints**. We’ve already seen two examples of them in (3.6) and (3.7). Note that we do not need to consider subsets S , where $S = \{i, j\}$, since this would give the redundant constraint

$$x_{ij} \leq 1.$$

By the same token, we do not need to consider subsets S , where $|S| \geq |V| - 2$. The complete model for the symmetric TSP is given in Integer Program 3.6.

Integer Program 3.6 IP Formulation of Traveling Salesperson

Problem

$$\min \sum_i \sum_{j>i} d_{ij} x_{ij}$$

s.t.

$$\sum_{i<j} x_{ij} + \sum_{i>j} x_{ji} = 2, \quad i \in V$$

$$\sum_{\substack{i,j \in S \\ (i,j) \in E}} x_{ij} \leq |S| - 1, \quad S \subset V, \quad 3 \leq |S| \leq |V| - 3$$

$$x_{ij} \in \{0, 1\}$$

■ EXAMPLE 3.10

Returning to Example 3.9, the complete optimization model would be

$$\begin{aligned} \min \quad & 10x_{12} + x_{13} + x_{15} + x_{24} + x_{26} \\ & + 10x_{34} + x_{35} + x_{46} + 10x_{56} \end{aligned}$$

s.t.

$$x_{12} + x_{13} + x_{15} = 2$$

$$x_{12} + x_{24} + x_{26} = 2$$

$$x_{13} + x_{34} + x_{35} = 2$$

$$x_{24} + x_{34} + x_{46} = 2$$

$$x_{15} + x_{35} + x_{56} = 2$$

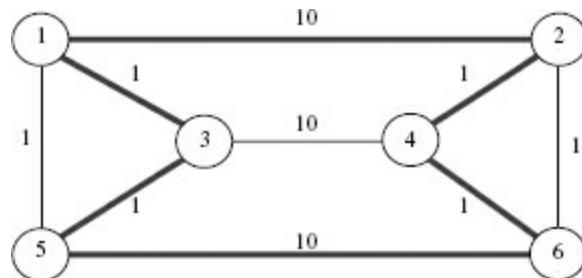
$$x_{26} + x_{46} + x_{56} = 2$$

$$x_{13} + x_{15} + x_{35} \leq 2$$

$$x_{24} + x_{26} + x_{46} \leq 2$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in E.$$

FIGURE 3.6 Optimal solution to symmetric TSP in Example 3.10.



Note that every other subset S consisting of exactly three vertices contains at

most two edges/variables, so their subtour elimination constraints (3.8) would be redundant. A solution to this integer program is given graphically in [Figure 3.6](#).

There is a problem, though. There could potentially be an exponential number of subtour elimination constraints, and we may not have enough time to write them, nor the space to put them into our optimization package. So how do we solve this? Similar to how we solved minimum spanning trees, we iteratively include those subtour elimination constraints that are not satisfied once a potential solution is found. For example, in Example 3.9, we would solve the problem without any subtour elimination constraints. If this solution gives a complete tour, we are done. If not, then some subtour elimination constraint must be violated. Once one has been identified, we add it to the problem and resolve. We continue to do this until a complete tour is derived.

Remark There are significant drawbacks to this solution approach. First, integer programs can be extremely difficult to solve to optimality, requiring plenty of computational time and resources. Linear programs are much easier to solve computationally. Second, identifying subtours can be difficult. Another approach, albeit similar in nature to the first, is to first solve the problem not as an integer program, but as a linear program by replacing the constraints $x_{ij} \in \{0, 1\}$ with the constraints/bounds $0 \leq x_{ij} \leq 1$. After each iteration, we check to see if (a) we have an integer solution and (b) if it is a complete tour. If it is not a complete tour, some subtour elimination constraint may be violated, and we add it to our model. If our solution is not integer, then we need to look for ways to force the variables to be integer without actually specifying that they be integer. These techniques, known as **cutting plane algorithms**, will be discussed in Chapter 14.

There is another way to eliminate subtours in solutions that differs from (3.8). First, we redefine x_{ij} to indicate whether the tour travels from node i to node j ; thus, there are variables x_{ij} and x_{ji} for each edge (i, j) of the graph. In addition, constraints (3.5) are rewritten, for each vertex i , as

$$(3.9) \quad \sum_{j \in V - \{i\}} x_{ij} = 1$$

$$(3.10) \quad \sum_{j \in V - \{i\}} x_{ji} = 1.$$

With each vertex i , we associate a *potential* u_i , which indicates a sense of

distance from node 1. Assuming that each u_i is nonnegative, we have the notion that if the tour travels from i to j , then we should have $u_j = u_i + 1$. In general, for any edge (i, j) in our graph, we will have either $u_j \geq u_i + 1$ or $u_j \leq u_i + 1$, and that if we have $x_{ij} = 1$ in our solution (so that the tour travels from i directly to j), we would have $u_j \geq u_i + 1$ satisfied. To model this, consider the constraint

$$(3.11) \quad u_j \geq u_i + 1 - M(1 - x_{ij}).$$

It is easy to see that if $x_{ij} = 1$, then the potentials u_k are oriented correctly, and that if $x_{ij} = 0$, we have a constraint that is always satisfied for large values of M . In fact, we can choose $M = n - 1$, provided that each label satisfies $1 \leq u_i \leq n$.

But, does this eliminate cycles from our solution? Suppose we had a solution like that given in [Figure 3.5](#) and each constraint (3.11) was satisfied. Since we could have $x_{26} = x_{64} = x_{42} = 1$ as a potential solution (or we could have had $x_{62} = x_{24} = x_{46} = 1$), this would lead to the satisfied constraints

$$\begin{aligned} u_6 &\geq u_2 + 1 \\ u_2 &\geq u_4 + 1 \\ u_4 &\geq u_6 + 1, \end{aligned}$$

which implies that $0 \geq 3$. In addition, to make sure that every tour contains every node, we fix one of the values of u_i ; typically, we fix $u_1 = 1$ and not include any of the constraints (3.11) involving vertex 1. Thus, if we have constraints (3.11) satisfied by variables u_i , and we have constraints (3.9) and (3.10) satisfied by the binary variables x_{ij} , we will not have any subtours in our problem. Thus, an alternative formulation of the traveling salesperson problem is

$$\begin{aligned}
\min \quad & \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j \in V - \{i\}} x_{ij} = 1, \quad i \in V \\
& \sum_{j \in V - \{i\}} x_{ji} = 1, \quad i \in V \\
& u_j \geq u_i + 1 - (n - 1)(1 - x_{ij}), \quad i \neq 1, j \neq 1 \\
& x_{ij} \in \{0, 1\} \\
& u_1 = 1 \\
(3.12) \quad & 2 \leq u_i \leq n, \quad i \neq 1.
\end{aligned}$$

This formulation adds only n variables and roughly $\frac{1}{2}n^2$ constraints to (3.9)–(3.10), making the formulation much more compact to write. This formulation was first introduced by Miller, Tucker, and Zemlin [65] and is referred to as the **Miller–Tucker–Zemlin (MTZ) formulation**.

So why introduce the first formulation? It turns out that the MTZ formulation can solve only problems with small number of vertices n without much computational difficulty; once problems become “large,” this formulation can take too long to solve. That being said, it is nice to see that there are multiple ways to model the same construct.

3.5 SPORTS SCHEDULING AND AN INTRODUCTION TO IP SOLUTION TECHNIQUES

So far, we have primarily been concerned with setting up an integer model and not too worried about how to solve it; we just let the computer do the work for us. While this is normally fine, there are times (and more often than you may think) where knowing how an integer program is solved helps us to produce an appropriate model. To see this, let’s consider the following problem.¹

■ EXAMPLE 3.11

The local YMCA is organizing a winter basketball league. Currently, they have 10 teams signed up. They would like to divide the teams into two divisions and play a “generalized round-robin,” where each team plays the others in their division X times and the teams in the other division Y times. Thus, they play a total of $4X + 5Y$ games. Since all games are played in the same gym, there is no need to designate one team as “home” and the other “away.” In addition, there is enough room so that all 10 teams can play on the same night. They would also like to have divisional games played later in the season, so that the appropriate playoff teams can be best determined. They want to examine the possible schedule that can be generated for (X, Y) values of $(1, 1), (2, 1), (3, 1)$, and $(3, 2)$.

To model this situation, let T be the set of teams, and for simplicity, we shall assume that teams 1, 2, 3, 4, and 5 are in one division and teams 6, 7, 8, 9, and 10 are in the other division. We can define variables x_{ijk} as

$$x_{ijk} = \begin{cases} 1, & \text{if team } i \text{ plays team } j \text{ in time slot } k, \\ 0, & \text{otherwise.} \end{cases}$$

We shall define these variables only for $i < j$ to avoid duplication. Note that $i, j \in T$ and $k \in W = \{1, 2, \dots, 4X + 5Y\}$ (here, W denotes the set of time slots). There are two major types of constraints:

1. Each pair of teams must play the correct number of games.
2. No team can play more than one game per time slot.

These can be modeled by the following constraints:

$$(1) \sum_{k \in W} x_{ijk} = \begin{cases} X, & \text{if } i, j \text{ are in the same division,} \\ Y, & \text{if } i, j \text{ are in different divisions,} \end{cases} \quad i < j, i, j \in T$$

$$(2) \sum_{j:j < i} x_{jik} + \sum_{j:j > i} x_{ijk} = 1, \quad i \in T, k \in W.$$

For an objective function, one way to model the situation that divisional games are played later in the season is to place greater weight on late-season divisional games and maximize our function. One way to do this is to let

$$c_{ijk} = \begin{cases} 0, & \text{if } i, j \text{ are in opposite divisions,} \\ 2^{k-1}, & \text{if } i, j \text{ are in the same division.} \end{cases}$$

Note that this places much greater emphasis on late-season divisional games. Finally, we need the binary restrictions $x_{ijk} \in \{0, 1\}$. For this model, there are $\binom{10}{2}(4X + 5Y)$ variables and $\binom{10}{2}(4X + 5Y)$ constraints.

Clearly, at this point, the set of constraints we've defined above seems minimal. Often in such leagues, especially ones where there are teams that play every year, the league pretty well knows ahead of time which teams are expected to be strong. A good league manager may want to have these teams play as late in the season as possible because of the playoff implications it creates. Also, we have ignored constraints that says two teams cannot play each other on back-to-back time slots. We shall leave such constraints for the reader to consider.

Let's try to solve our problem using our modeling software. (Note: Due to the size of these models, you may have too many variables for the student version of your software.) We use the solver Xpress-MP Release 2008A. Trying the $(X, Y) = (1, 1)$ case first, we see that an optimal solution is found quickly. Even the $(X, Y) = (2, 1)$ and $(3, 1)$ cases solve fairly easily, although they take slightly longer than the first case. However, if we try the $(3, 2)$ case, it takes much longer to solve. An important question to ask is WHY?

To answer this question, we first need to understand how integer programs are generally solved. We will be going into much greater detail in Chapter 14, but for now the basics are needed. Nearly every commercial integer programming optimization package uses a technique known as *branch-and-bound* to solve them. In a nutshell, branch-and-bound systematically partitions the feasible region of the underlying linear program (the one where the integrality constraints are removed), generating many smaller regions in which it tries to find the best integer solution. One important aspect of this approach is that if there are a lot of "excess" fractional solutions, the algorithm takes longer to find the best integer solution. Thus, as modelers we often try to minimize these excess solutions.

But how do we do this? Let's consider the $(3, 2)$ case. If we remove the integrality restrictions on x_{ijk} , we find that in time slot $k = 6$, $x_{1,2,6} = x_{1,3,6} = x_{2,3,6} = 0.5$, $x_{6,7,6} = x_{6,10,6} = x_{7,8,6} = x_{8,9,6} = x_{9,10,6} = 0.5$, and $x_{4,5,17} = 1$. This is a valid solution to our constraints. However, logically we know that with five teams in the division, no more than two divisional games can be played this time slot, and we are being told to schedule 2.5 games. This translates to the constraint

$$\sum_{i,j \in \{1,2,3,4,5\}} x_{i,j,6} \leq 2,$$

which our current solution does not satisfy. Thus, this solution is an excess solution. We call such a constraint a *cut*. In general, a cut is a constraint in which (1) every feasible integer solution satisfies it, and (2) there are fractional solutions to our original problem (with integrality constraints removed) that do not satisfy it. The general forms for these cuts are, for each time slot k ,

$$\sum_{i,j \in \{1,2,3,4,5\}} x_{i,j,k} \leq 2$$

$$\sum_{i,j \in \{6,7,8,9,10\}} x_{i,j,k} \leq 2.$$

If we add these cuts to our problem, because they eliminate many excess fractional solutions, the hope is that our algorithm will find an integer optimal solution more quickly. In fact, if we add all such cuts to our (3, 2) problem, we find that it solves very quickly.

This provides an early example of the difficulty that lies ahead when solving integer programs. When modeling integer programs, we often have to consider how easily it can be solved; because integer programs are much more difficult to solve computationally, we often need to alter our model to have some software package solve it in a “reasonable” amount of time. We will explore these ideas further in Chapters 13 and 14.

Summary

In this chapter, we explored various models where binary variables are explicitly used within the model, typically in the formulation of constraints. Such approaches are often used when modeling real-world situations. It’s probably safe to say that binary variables and their applications are one of the most widely used modeling constructs in operations research. The exercises in this chapter explore some other uses of binary variables, while Chapter 4 will introduce more examples, including some models that are commonly used in business and industry.

EXERCISES

3.1 Your company produces breakfast cereal at five different plants. The monthly capacity (in tons) of each plant is given below. The cereal is

stored at one of the three warehouses. The per ton cost of producing cereal at each plant and shipping it to each warehouse is given below. You have four customers. The cost of shipping a ton of cereal from each warehouse to each customer is also given below. Each customer must receive the amount (in tons) of cereal given below. The annual fixed cost of operating each plant and warehouse is also given below. Your goal is to minimize the annual cost of meeting customer demands. Which plants and warehouses should be open, and what is the optimal shipping plan?

		Capacities (tons)				
		1	2	3	4	5
Plant		300	200	300	200	400
Customers						
		1	2	3	4	
Requirements (tons)		200	300	150	250	

Shipping Costs								
		To Warehouses			To Customers			Fixed Cost
		1	2	3	1	2	3	
From	Plant 1	\$800	\$1000	\$1200	—	—	—	\$35,000
	Plant 2	\$700	\$500	\$700	—	—	—	\$45,000
	Plant 3	\$800	\$600	\$500	—	—	—	\$40,000
	Plant 4	\$500	\$600	\$700	—	—	—	\$42,000
	Plant 5	\$700	\$600	\$500	—	—	—	\$40,000
From	Warehouse 1	—	—	—	\$40	\$80	\$90	\$50
	Warehouse 2	—	—	—	\$70	\$40	\$60	\$80
	Warehouse 3	—	—	—	\$80	\$30	\$50	\$60

3.2 At a machine tool plant, five jobs must be completed each day. The time it takes to do each job depends on the machine used to do the job, but not every machine can handle every job. If a machine is used at all, there is a (one-time only) setup time required. The relevant times (in minutes) are given below.

	Job 1	Job 2	Job 3	Job 4	Job 5	Setup Time
Machine 1	35	47	63	27	—	40
Machine 2	—	50	45	—	40	50
Machine 3	38	42	—	37	—	50
Machine 4	43	—	55	—	38	60
Machine 5	—	40	—	54	32	60

Each machine can handle at most two jobs per day, and in addition, since machines 1 and 2 require the same people to do setup, an additional 20 minutes is needed if both machines are used. Determine how to minimize

the sum of the setup and machine operation times needed to complete all jobs.

3.3 Three different products are made on three production lines each week. If a production line is used in a given week there is an associated setup cost. Each worker is designated to only one production line, and the pay and production of each worker depends on which line they are assigned to. In addition, each worker is assigned to one product on their assigned line. Relevant data are given below.

	Line 1	Line 2	Line 3
Setup cost	\$2000	\$3000	\$4000
Product 1/worker	50	90	120
Product 2/worker	75	110	130
Product 3/worker	90	125	150
Cost/worker	\$700	\$1000	\$1500

Each week we need to make 600 of product 1, 800 of product 2, and 1000 of product 3. If we can use at most 20 workers, formulate and solve an integer program that minimizes the total cost of meeting our weekly demand.

3.4 A job requires 10 operations to be done on any of three machines. The time taken for each operation on each of the given machines is given below.

	Operation Times									
	1	2	3	4	5	6	7	8	9	10
Machine 1	5	9	2	3	8	6	4	9	8	5
Machine 2	6	7	4	4	6	5	6	5	3	8
Machine 3	4	10	3	4	5	7	7	6	5	4

Furthermore, jobs 2 and 6 must be completed on the same machine. The total time required to do all 10 operations is the maximum time used by the three machines. Formulate and solve an integer program that minimizes the total time needed to do all the operations.

3.5 Spider Airlines has decided that it needs to add hubs in the United States. Spider runs flights between the following cities: Atlanta, Boston, Chicago, Dallas, Los Angeles, New York, Oklahoma City, Pittsburgh, Richmond, Salt Lake City, San Francisco, and Seattle. It needs to have a hub within 1200 miles of each of these cities. The company wants to determine the smallest number of hubs needed to meet this requirement.

	Cities Within 1200 Miles
Atlanta (AT)	AT, CH, DA, OC, NY, PI, RI
Boston (BO)	BO, NY, PI, RI
Chicago (CH)	AT, BO, CH, DA, NY, OC, PI, RI
Dallas (DA)	AT, CH, DA, OC
Los Angeles (LA)	LA, SL, SF, SE
New York (NY)	AT, BO, CH, NY, PI, RI
Oklahoma City (OC)	AT, CH, DA, OC, SL
Pittsburgh (PI)	AT, BO, CH, NY, PI, RI
Richmond (RI)	AT, BO, CH, NY, PI, RI
Salt Lake City (SL)	LA, OC, SL, SF, SE
San Francisco (SF)	LA, SL, SF, SE
Seattle (SE)	LA, SL, SF, SE

3.6 A local bakery is planning the expansion of its bread-making capacity for the next 5 months. Currently, it can make 5000 loaves per month, but it forecasts its monthly demands to be as given in the following table.

	Month 1	Month 2	Month 3	Month 4	Month 5
Demand	5500	6500	8000	8500	9500

It can increase its current capacity by installing one of the three sized ovens. Each has a one-time purchase cost and a monthly usage cost as given below.

Oven Type	Monthly Capacity	Purchase Cost	Monthly Usage Cost
Small	1000	\$300	\$75
Medium	2000	\$500	\$100
Large	3500	\$1000	\$125

An oven can be used to full capacity during the month it is purchased. Formulate and solve an integer program that determines when an oven is bought, if we are interested in minimizing the total cost.

3.7 (Based on Boykin [20]) A chemical production company annually produces 500 million pounds of the chemical maleic anhydride using four different reactors. Each reactor can be run on only one of the four settings. The following table gives the annual cost (thousands of dollars) and production (in millions of pounds) for each reactor and each setting.

	Setting	Cost	Pounds
Reactor 1	1	70	75
	2	90	120
	3	120	150
	4	160	205
	Setting	Cost	Pounds
Reactor 2	1	55	60
	2	80	100
	3	110	135
	4	150	190
Reactor 3	1	80	90
	2	100	140
	3	140	175
	4	180	225
Reactor 4	1	40	50
	2	70	90
	3	90	110
	4	110	150

A reactor can only be run on one setting for the entire year. How can they meet its annual demand at a minimum cost?

3.8 Spider's Sports Bar is updating all its televisions to better serve its customers. It is considering the purchase of both 100-inch LCDs costing \$5000 to place on large walls and smaller 32-inch flat screens costing \$750 to hang from the ceiling. They want to purchase at least two 100-inch TVs so that almost everyone in the bar can view a game on them, but they also want to purchase at least eight smaller TVs so that different games can be shown. It is important to the owners that everyone in the bar can view at least two TVs; if customers cannot see a 100-inch TV optimally from their position, then they need to see three TVs. To help obtain the locations for these TVs, the bar was partitioned into 12 zones, and for each zone the optimally viewed TV locations were determined from among the potential locations of the 100-inch TVs (labeled A, B, C, D, E, and F) and the 32-inch locations (labeled 1–22); this information is given in the following table.

Zone	TV Locations	Zone	TV Locations
1	1,2,3,C, D	7	A, B 7, 14, 15, 16
2	3, 4, C, D	8	A, B, F 8, 10
3	A, C, 4, 5, 13	9	A, B, F, 9, 16, 17
4	A, C, D, 4, 6	10	E, F, 20, 21, 22
5	A, C, 5, 7, 13, 14	11	B, E, F, 19, 20
6	A, B, 6, 7, 8	12	B, F, 11, 18, 19

Formulate and solve an integer program to determine the locations of the various TVs so as to minimize total cost.

3.9 Referring back to Exercise 2.13, suppose that if a given alloy is used, at least 10 tons of it must be used, and that if a scrap piece is used, at least 25 tons of it must be used. How can you now produce the required 100-ton steel at minimum cost?

3.10 In the manufacturing of printed circuit boards, holes need to be drilled on the boards through which chips and other components are later wired. It is required that these holes be drilled as quickly as possible; hence, the problem of the most efficient order to drill the holes is a traveling salesperson problem. In the table below are the distances (in millimeters) between any pair of hole locations. Hence, minimum time equals finding the minimum total distance traveled between the hole locations.

Determine the minimum time required to drill all the needed holes using the following models:

- (a) Integer Program 3.6 (using subtour elimination constraints).
- (b) The MTZ model.

Hole Locations	1	2	3	4	5	6	7	8
1	—	13	8	15	7	16	19	21
2	13	—	5	7	14	22	11	14
3	8	5	—	15	17	9	13	12
4	15	7	15	—	8	7	9	10
5	7	14	17	8	—	12	18	11
6	16	22	9	7	12	—	8	14
7	19	11	13	9	18	8	—	15
8	21	14	12	10	11	14	15	—

3.11 Often, a tooling machine is outfitted with different tools during a day to help produce different parts. There is a delay incurred during the changing from one tool to another. Suppose your tooling machine has eight different tools that are used during a given day. The matrix below gives the delay time t_{ij} (in minutes) of changing from tool i to tool j .

Delay Times	1	2	3	4	5	6	7	8
1	—	13	8	15	7	16	19	21
2	10	—	5	7	14	22	11	14
3	6	8	—	15	17	9	13	12
4	12	16	12	—	8	7	9	10
5	18	5	11	14	—	12	18	11
6	9	13	18	7	6	—	8	14
7	20	10	7	6	11	15	—	15
8	13	14	11	17	18	8	11	—

Assuming that the order used during one day's operation is the same for the next day, and that once the final tool is used the operators must reinstall the first tool used before their shift ends, find the order the tools should be used on the machine in order to minimize the total changing time.

3.12 Formulate and solve the TSP in Example 3.8 using the MTZ model.

3.13 A gasoline delivery truck contains five compartments, holding up to 2500, 3000, 1400, 1600, and 3200 gallons of fuel, respectively. The company must deliver three types of fuel (super, premium, and regular) to a customer. The demands, penalty per gallon short, and the maximum allowed shortage are given below. Each compartment of the truck can carry only one type of gasoline. How should the company load the truck in order to minimize shortage costs?

Type of Gasoline	Demand	\$ Per Gallon Short	Max Allowed Shortage
Super	2800	10	500
Premium	4200	6	400
Regular	5000	8	300

3.14 Sunshine Rental Properties rents beachfront properties to vacationers in Florida throughout the year. They currently have purchased six additional properties that are in various states of disrepair. For each property they have determined the repair time, number of workers needed for the repair, and the monthly rent they can get once the repairs are done. This information is given in the table below.

	Property					
	1	2	3	4	5	6
Repair time (months)	2	3	2	1	3	2
Workers required	10	5	8	10	6	7
Monthly rent (\$)	4000	7500	3000	2500	7000	3500

Once a repair job has been started it cannot be stopped until completion and those workers must stay on that job. Sunshine has 15 people available

to work each month. Formulate and solve an integer program that determines the repair schedule for the next year in order to maximize Sunshine's acquired rent.

3.15 Sudoku is a logic puzzle that has recently become very popular worldwide. In it, you are given a 9×9 grid with some numbers from 1 to 9 in various squares. To solve the puzzle, you must fill in the remaining squares so that, in each row, column, and 3×3 block, each number from 1 to 9 appears exactly once. Below is a sample puzzle.

9		4	3	7			1	
						4		
	3						8	7
					1			
	2	7		3		1	5	
			5		6			4
7	8		6					9
		6						
	9		4	7				3

Formulate an integer program that will solve the given puzzle. (*Note:* You may not be able to solve this model using your student edition software!)

3.16 In workshops it frequently happens that a single machine determines the throughput of the entire production (e.g., a machine of which only one is available, the slowest machine in the production line, etc.). In this case, a set of jobs is to be processed on a single machine, and careful scheduling of these jobs is a high priority for the workshop. We shall assume that once a job has been started, its operation cannot be interrupted until it is completed. Below are the relevant data for a workshop where seven jobs are to be scheduled for a given machine; for every job, its release date and duration are given, as well as its due date (desired latest completion time).

Job	1	2	3	4	5	6	7
Release date	1	5	9	0	0	8	11
Duration	6	7	10	6	2	5	5
Due date	10	15	22	11	5	19	22

How can we schedule the jobs on this machine in order to minimize the total duration of the schedule (known as the makespan)?

3.17 To see the effect large values of M can have, resolve Integer Program

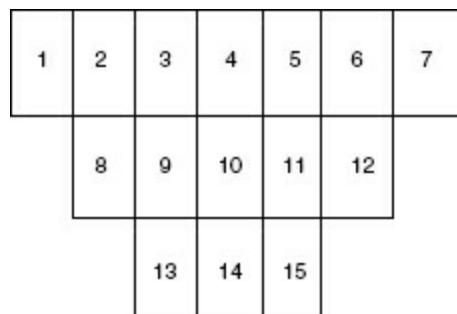
3.4 using the following values of M : $10^2, 10^4, 10^8, 10^{10}, 10^{12}$.

3.18 Consider the data given in Exercise 3.16. The tardiness of a job is defined to be 0 if it is completed before or at its due date and (completion time – due date) otherwise. How can we schedule the jobs so as to minimize the total tardiness (sum of the tardiness over all jobs)?

3.19 Before leaving for the holiday you wish to backup your most important files onto CDs. These files have sizes 240, 462, 117, 560, 379, 110, 341, 294, 503, 469, 90, 63, 617, 493, 524, and 396 MB. If the capacity of a CD is 780 MB, formulate and solve a model to determine the minimum number of CDs needed to save all the files.

3.20 In mining operations, one of the most important problems is to determine the best contour of an open-pit mine. In an open-pit mine, the mined region is divided into blocks on various levels. Depending on the geography of the mine and the technology used, restrictions are imposed on how the blocks can be accessed. For example, consider the mine given in [Figure 3.7](#) consisting of three levels. In this setup, a block cannot be removed and accessed unless the blocks immediately above it and to its right and left are removed; for example, block 10 cannot be accessed unless blocks 3, 4, and 5 are removed, while block 13 requires blocks 8, 9, and 10 to be removed. Suppose that, in the example above, we are trying to extract a precious ore from this mine. Based on the results of test drillings, it has been determined that blocks 3, 5, 9, 10, 12, and 15 contain the desired ore. The amount of ore in each of these blocks varies, so that their market value (per ton) is given in the following table.

FIGURE 3.7 Open-pit mine example.



Block	3	5	9	10	12	15
Value of ore (per ton)	300	800	600	1000	600	1300

If it costs \$200 per ton to remove a block in the top level, \$350 per ton in the middle level, and \$600 per ton in the bottom level, which blocks should be extracted to maximize the profit margin?

3.21 Your company produces three different products that are each produced on four different machines. However, each product uses the machines in different orders and for different durations (minutes). Below are the relevant data.

Product 1		Product 2		Product 3	
Machine Order	Duration	Machine Order	Duration	Machine Order	Duration
1	3	3	5	4	2
4	7	1	10	2	15
3	12	2	9	1	7
2	4	4	6	3	3

How should the products be scheduled on the machines in order to finish the production of each product as early as possible? Indicate the starting time for each product on each machine.

3.22 Until now, your company has been sending its product to customers out of one depot. However, you have started to grow so fast that it determines that it should open new depot centers to distribute its goods. After some research, potential depot sites have been identified. Every new depot has a one-time fixed cost consisting of construction and other capital costs. In addition, given the various locales, each potential depot has a different weekly production capacity limit. The data are given in the table below.

Depot	1	2	3	4	5	6
Cost (\$1000)	35,000	40,000	28,000	37,000	45,000	50,000
Capacity (tons)	200	240	180	210	270	390

Each customer's demand, which is given below, will be met by only one depot.

Customer	1	2	3	4	5	6	7	8	9	10	11	12
Demand (tons)	80	40	50	65	90	30	75	20	45	60	40	55

Delivery costs (per ton) of shipping goods from a depot to a customer are also given below. Note that, if there is no amount given, your company has already determined that that depot will not deliver goods to that company.

(Per Ton) Delivery Costs for Shipping Goods from Depot to Customer												
Depot	Customer											
	1	2	3	4	5	6	7	8	9	10	11	12
1	20	35	48	—	—	—	37	22	18	30	45	35
2	27	55	62	45	20	18	—	—	15	32	27	20
3	—	—	25	30	45	60	15	50	40	30	40	50
4	—	25	35	50	40	20	15	30	35	—	—	25
5	45	—	—	—	35	50	40	20	30	15	20	—
6	30	30	40	45	—	55	40	30	45	30	—	40

Which depots should be opened in order to satisfy all customer demands at the minimum cost?

3.23 In Exercise 2.25, we were asked to determine the time required to finish a project consisting of 11 tasks with varying duration times. Unfortunately, not all tasks can be done at the same time (although some can be done in parallel); certain predecessor tasks must be completed before others can be started. The table below gives the duration and predecessors for each task. Thus, for example, task J cannot begin before each of tasks F, G, and H are completed.

Task	Duration	Immediate Predecessors
A	2	None
B	3	A
C	5	B
D	7	B
E	3	B
F	4	D
G	2	E
H	6	E
I	8	C, F
J	6	F, G, H
K	9	I, J

TABLE 3.1 Arrival Times and Penalties for Each Plane

	Planes									
	1	2	3	4	5	6	7	8	9	10
Earliest time	45	30	120	100	90	75	140	145	130	110
Target time	75	65	160	125	120	110	200	180	175	150
Latest time	150	150	220	200	180	190	275	250	260	225
Early penalty	25	25	45	50	40	40	50	45	50	35
Late penalty	50	50	70	75	90	100	85	65	65	60

Suppose these tasks are to be done on three workstations on an assembly line. Each task needs to be assigned to a single workstation that completes it without interruption. The tasks must be assigned to the workstations so that each succeeding task is done on either the current workstation or one of the successor workstations; for example, if task D is on workstation 2, then task F (which cannot begin until D is finished) must be assigned to either workstation 2 or 3 but not 1. Formulate and solve an integer program that assigns tasks to workstations so that the precedence ordering is maintained and the maximum time allocated to a workstation is minimized.

3.24 (Based on Beasley [12]) A small regional airport is expecting 10 planes to land on its lone runway used for arriving planes. Each aircraft has indicated a target time, but has also indicated an earliest possible arrival (if it flies at maximum speed) and latest possible arrival time (if it flies at its most fuel-efficient speed and circles above airport for maximum time). Ideally, each aircraft would arrive at its target time, but this rarely occurs. Thus, in order to schedule the various landings, the airport penalizes an aircraft if it arrives earlier or it arrives later than its target time. The times (given in minutes after 9 AM) and penalties for each plane are given in [Table 3.1](#).

Because of possible air turbulence concerns, there is a minimum amount of time between the landing of any two aircraft, and these times are given in [Table 3.2](#). These separation times are required regardless of when they land; thus, the planes need not land successively for these times to be enforced.

TABLE 3.2 Separation Times Between Plane Landings

	1	2	3	4	5	6	7	8	9	10
1	—	5	10	8	4	7	10	6	10	7
2	5	—	15	15	10	8	10	15	10	15
3	10	15	—	10	5	15	10	7	15	10
4	8	15	10	—	7	10	5	8	8	10
5	4	10	5	7	—	10	15	10	15	8
6	7	8	15	10	10	—	8	10	6	9
7	10	10	15	8	15	6	—	10	10	8
8	6	15	7	8	10	10	10	—	8	10
9	10	10	15	8	15	6	10	8	—	7
10	7	15	10	10	8	9	8	10	7	—

Formulate and solve an integer program that determines the minimum cost

landing schedule.

3.25 Alter and resolve your model from Exercise 3.24 if you have two runways available.

3.26 (Based on Gicquel et al. [41]) Glass coating involves depositing thin layers of distinct metals onto different layers in order to attain different properties such as thermal insulation. Different sequences of metal layers affect the desired properties. Coating can be done using lines of metallic cathodes that spray a specific metal onto the glass sheet, where each cathode c has a maximum amount V_c of metal available. For example, if we have two products that require the metal sequences A, B, C and B, A, C, then we would need at least four cathodes in the sequence A, B, A, C to produce both products; in the first product, the third cathode does not spray anything, while the first cathode does not spray the second product. The problem for the glass industry is to determine the order of cathodes and spraying sequence in order to produce the desired number of different products while minimizing the number of cathodes used.

Suppose we have four different metals used to produce three products requiring three layers each. The ordering of metal layers (and each layer's volume) for the three products is given below.

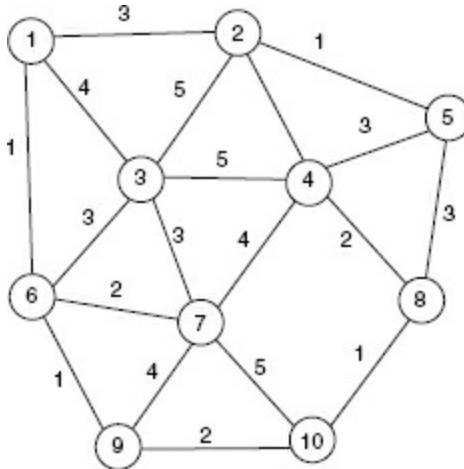
	Layer		
	1	2	3
Product 1	Metal	A	B
	Amount	400	400
Product 2	Metal	D	C
	Amount	500	700
Product 3	Metal	C	D
	Amount	900	1000
			A
			100
			800

The products are produced one at a time in the above order. In addition, the following four cathode types are available, with their available quantity and metal amount given below.

Cathode Metal	Quantity	Maximum Amount of Metal
A	3	1000
B	4	1500
C	3	1200
D	3	1200

Formulate and solve an integer program that determines a sequence of cathodes and a spraying of metal to each product that minimizes the total number of cathodes used.

FIGURE 3.8 Tower configuration for Exercise 3.27.



3.27 Due to the explosion of wireless communication activities available today, companies have been working to determine available frequencies that can be assigned to wireless devices in a geographic area so that no interference between nearby devices is possible. Suppose a communications company has 10 towers in a small region, which are denoted in [Figure 3.8](#) as nodes of a graph $G = (V, E)$, while the edges correspond to those towers for which frequencies may interfere. Suppose that each tower is to receive two frequencies, which are represented by positive integers. The frequencies at each tower must differ by at least 10, while for each $(i, j) \in E$ each frequency assigned to i must differ by at least c_{ij} from each frequency of j , where c_{ij} is the weight assigned to edge $c_{i,j}$. Formulate and solve an integer program that minimizes the span of frequencies used.

3.28 In Exercise 2.1, we saw that CT Furniture Company makes wooden picnic tables and chairs to sell to various stores. These stores request both finished and unfinished furniture. To make this furniture, CT must purchase the wood by the board foot. It takes 25 board feet to make one table and 10 board feet to make one chair. At most 10,000 board feet can be purchased, and a quantity discount is available; the first 4000 boards cost \$2 per foot, and the next 3000 boards cost \$1.50 per foot, while the

remaining 3000 boards cost \$1 per foot. It takes 4 hours to make an unfinished table and 2 hours to make an unfinished chair. To finish an unfinished table (by priming, sealing, and painting it), it takes an additional 8 hours, while it takes 6 hours to finish an unfinished chair. A total of 2500 hours are available, which can be proportioned as necessary. We can assume that all pieces of furniture produced can be sold at the following prices: \$80 for unfinished table, \$120 for finished table, \$30 for unfinished chair, and \$55 for finished chair. In addition, due to typical demand, we know that, for every table produced, at least two chairs must be produced. If we have special orders that must be filled, calling for at least 200 tables and 450 chairs to be sold, determine how much of each type of furniture to produce in order to maximize CT's profit.

3.29 (Based on Katok and Ott [62]) Hoosier Can produces aluminum cans for a local beverage company. This requires the production of cans with distinct labels for each of the four beverages produced. There are three production lines available, which cover the 14 12-hour shifts each week. For each label and production line, the number of cans/labels produced per minute on that line is given in the following table.

	Time Per Label (s)			Cost Per Label (\$)		
	Line 1	Line 2	Line 3	Line 1	Line 2	Line 3
Label 1	0.20	0.30	0.10	0.15	0.20	0.25
Label 2	0.35	0.20	0.15	0.20	0.30	0.15
Label 3	0.25	0.30	0.10	0.25	0.15	0.20

During each shift at most one label can be produced on any given line, and the time (in hours) and cost associated with switching a line to a label is given below.

	Time to Switch (h)			Cost to Switch (\$)		
	Line 1	Line 2	Line 3	Line 1	Line 2	Line 3
Label 1	1.0	1.0	1.2	75	100	120
Label 2	0.8	1.2	1.0	60	150	90
Label 3	1.2	0.7	0.9	135	100	115

The weekly demand for each label is found in the following table.

	Label 1	Label 2	Label 3
Demand	1,000,000	2,000,000	5,000,000

Formulate and solve an integer program that determines the production schedule for a given week that minimizes the total cost to Hoosier Can.

3.30 We saw that, for the traveling salesperson problem, we can construct subtour elimination constraints via the MTZ formulation. Construct a similar formulation for the minimum spanning tree problem.

3.31 Show that, for the symmetric TSP, you can also write the subtour elimination constraints (3.8) as

$$\sum_{i \in S} \sum_{j \notin S, j > i} x_{ij} + \sum_{i \notin S} \sum_{j \in S, j > i} x_{ij} \geq 2,$$

forall $S \subset N$, $|S| \geq 3$.

3.32 Suppose that you are given an asymmetric TSP. In this case, legs between two cities are differentiated between directions, that is, legs $i \rightarrow j$ and $j \rightarrow i$ are different, and must be modeled so. How would you transform the symmetric TSP model (Integer Program 3.6) into one for the asymmetric TSP?

3.33 Suppose that the variable y can take only the values 0, 3, 7, 15. How can we model this using linear constraints? Additional variables are allowed.

3.34 Often in optimization modeling, there is some interaction between two decisions. For example, if we open two warehouses in our fixed-charge minimum cost network flow model, there could be a savings involved due to their proximity. This can be modeled in the objective function using the nonlinear term $Cx_i x_j$, where both x_i and x_j are 0–1 variables. Since we cannot handle nonlinear terms using what we know, we wish to model this using linear terms and constraints. One method is to create a new variable y_{ij} and constrain it somehow to satisfy the equation $y_{ij} = x_i x_j$ when all variables are integers. What constraints would you use to guarantee this relationship?

3.35 Let

$$X = \left\{ (x, y) : \begin{array}{l} 2x + y \leq 10 \\ x + y \leq 8 \\ x \leq 4 \\ x, y \geq 0, \text{ integer} \end{array} \right\},$$

and note that $(2, 3) \in X$. How would you formulate the (linear) constraints for $Y = X - \{(2, 3)\}$?

3.36 Model the logical condition

$$\sum_{j=1}^n a_j x_j \leq b \text{ if and only if } y = 1,$$

using linear inequalities, where y is a binary variable.

3.37 Model the logical condition

$$\sum_{j=1}^n a_j x_j = b \text{ if and only if } y = 1,$$

using linear inequalities, where y is a binary variable.

[1](#)The material in this section is motivated by Trick [83].

CHAPTER 4

REAL-WORLD OPERATIONS RESEARCH APPLICATIONS: AN INTRODUCTION

In the previous chapters we considered some basic modeling approaches that often appear. We now examine three classes of problems and one specific industrial application area that are very common operations research models solved many times in business and industry. These problems, the *Vehicle Routing*, *Facility Location*, and *Network Design* problems, represent some of the core OR models that have been explored because they arise in many different settings. In addition, we explore OR applications to the airline industry, which is one of the largest users of operations research techniques. Throughout this chapter we consider the general model and its formation and then apply it to a specific case, which is typically how such models are introduced in the research literature.

4.1 VEHICLE ROUTING PROBLEMS

In Section 3.4 we considered the traveling salesperson problem (TSP), where we sought a cycle through each node of a graph that has the minimum total edge weights. This problem arises, for example, when determining the shortest route of a delivery person to all possible locations. But what if multiple delivery people, all starting from the same location, were allowed? This results not in the TSP but in the **Vehicle Routing Problem (VRP)**.

In the basic version of the VRP, vehicles with the same capacity, based at a

single depot station, serve many different customers. Each customer's demand is delivered by exactly one vehicle, and our goal is to find the minimum cost collection of vehicle routes, all starting and ending at the same depot, that contain all customers and do not violate the vehicle capacities. Mathematically, the VRP is defined on an undirected graph $G = (V, E)$, where $V = \{0, 1, \dots, n\}$ is the set of nodes and E is the set of edges of the graph, where we assume that vertex 0 is the depot, while the remaining nodes are the customers. A fleet of m identical vehicles, each with capacity D , is based at the depot. Each customer i has a demand d_i associated with it, and there is a cost c_{ij} associated with a route traveling between locations i and j (customers and the depot). The VRP seeks to find a set of m vehicle routes such that (a) each route begins and ends at the depot 0; (b) each customer is included on exactly one route; (c) the total demand of each route does not exceed D , and (d) the total cost associated with each route is minimized.

■ EXAMPLE 4.1

A local pizza shop received 10 late orders for delivery last night; unfortunately, only three delivery persons are working. The shop uses a coordinate system to mark where houses are located (using the nearest intersection as locations). The 10 deliveries are to go to the following places:

	1	2	3	4	5	6	7	8	9	10
E/W	20	40	180	130	160	50	30	100	90	75
N/S	90	70	20	100	10	80	50	60	120	15

All streets in this town go either north-south or east-west, so distance must be measured rectilinearly. Assuming that the pizza shop is located at position $(0, 0)$ and that each driver can deliver at most five orders, how should the delivery routes be determined in order to minimize the total travel distance?

To model Example 4.1, we have the nodes $V = \{0, 1, \dots, 10\}$ and edges between every pair i, j of nodes. The travel cost c_{ij} is simply the rectilinear distance between the coordinates; for example, we would have

$$c_{26} = |20 - 160| + |90 - 10| = 220.$$

Each customer's demand would be $d_i = 1$ and each vehicle would have a capacity of $D = 5$.

We emulate our work on the traveling salesperson problem by defining

variables x_{ij} to be 1 if edge (i, j) is on some route and 0 otherwise; by symmetry we can assume that $i < j$. Our objective is to minimize the total tour length

$$\sum_{(i,j) \in E} c_{ij} x_{ij}.$$

For each customer $i \in V - \{0\}$, we would have the constraint

$$\sum_{j:i < j} x_{ij} + \sum_{j:j < i} x_{ji} = 2,$$

since each customer is on exactly one route that must enter and leave that customer's location. Since the pizza shop is included on $m = 3$ different routes, there should be $2m = 6$ edges active at the shop, giving the constraint

$$\sum_{j \in V - \{0\}} x_{0j} = 2m.$$

At this point, it is unclear how to handle the capacity constraints of the vehicles. We have a capacity constraint for each subset of customers, indicating how many can be on the same route. Thinking back to how we modeled the TSP, we may want to mimic the notion of subtour elimination constraints—generating only those that are violated at each iteration.

■ EXAMPLE 4.2

Considering the above discussion as it pertains to Example 4.1, we would solve the integer program

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:i < j} x_{ij} + \sum_{j:j < i} x_{ji} = 2, \quad i \in V - \{0\} \\ & \sum_{j \in \{1, \dots, 10\}} x_{0j} = 6 \\ & x_{ij} \in \{0, 1\}, \quad (i, j) \in E. \end{aligned}$$

Solving this problem yields the solution with routes $0 - 1 - 6 - 0$, $0 - 2 - 7 - 0$, and $0 - 5 - 3 - 10 - 0$, in addition to the subtour $4 - 8 - 9$; its value is 1100. Since we have the subtour $4 - 8 - 9$, we can use the subtour elimination constraint we saw for the TSP:

$$x_{48} + x_{49} + x_{89} \leq 2.$$

Adding this constraint to the problem and resolving, we get the solution with routes $0 - 1 - 6 - 0$, $0 - 2 - 7 - 0$, and $0 - 5 - 3 - 4 - 9 - 8 - 10 - 0$, with value 1120. This last route contains six customers, and we are capped at five per vehicle. What constraint can we add that would eliminate this route? If we tried a TSP-based subtour elimination constraint, we could actually get a scenario where actually adding a station to the route would seem feasible; for example, if we say no more than six edges among the nodes $\{0, 3, 4, 5, 8, 9, 10\}$ can have value 1, the route $0 - 5 - 3 - 4 - 9 - 8 - 10 - 2 - 0$ would satisfy this condition, but this seems to be going in the wrong direction. In fact, it seems that we should not consider the pizza shop in this constraint, but only the customers to visit. Considering the customers $\{3, 4, 5, 8, 9, 10\}$, not only would we want at most five of the edges between them to be active, but since this is already the case in the current solution, we would want at most four. This is because we need at least two vehicles to service these six customers. This leads us to the constraint

$$\sum_{\substack{i,j \in \{3,4,5,8,9,10\} \\ i < j}} x_{ij} \leq 6 - 2 = 4.$$

Adding this constraint and resolving yields the solution with routes $0 - 1 - 6 - 2 - 0$, $0 - 7 - 10 - 0$, and $0 - 5 - 3 - 4 - 9 - 8 - 0$, with value 1130. Since we have all customers on three existing routes, and each route satisfies the capacity requirements of the vehicles, this is the optimal solution.

We can generalize the subtour elimination constraints described in Example 4.2. Given a set of customers S , suppose that $l(S)$ is a lower bound on the number of vehicles needed to service each customer in S . The subtour elimination constraints of the TSP can then be generalized to the constraint

$$(4.1) \quad \sum_{\substack{i,j \in S \\ i < j}} x_{ij} \leq |S| - l(S).$$

This constraint eliminates not only those routes that do not satisfy the capacity constraints, but also those subtours that do not include the depot on their route. We saw this in Example 4.2, where a constraint of this form was used to eliminate the subtour involving 4, 8, 9 (here $l(\{4, 8, 9\}) = 1$) since all three can be in the same route) as well as removing from consideration routes that violated the capacity constraints. In fact, any restriction on the size of the routes can be indirectly modeled using (4.1). Thus, a formulation of the VRP

would be given in Integer Program 4.1.

This formulation assumes that every vehicle visits at least two customers, and that the direction a route is traversed does not change the cost (symmetric case). This model can be easily adapted to include the case where a vehicle may visit only one customer (see Exercise 4.17) and to the asymmetric case where the direction a route is traversed influences the total cost (see Exercise 4.18).

Many variations of this problem exist, including (but not limited to) models where (a) each customer must be visited only within a specified time window (VRP with time windows), (b) vehicles start from one of the multiple depots (multidepot VRP), and (c) customers can possibly be served by more than one vehicle (split delivery VRP). Recent books by Golden et al. [46] and Toth and Vigo [82] discuss the recent research on these and other variations of the problem.

Integer Program 4.1 Formulation of Vehicle Routing Problem

$$\begin{aligned}
 & \min \sum_{(i,j) \in E} c_{ij} x_{ij} \\
 \text{s.t.} \\
 & \sum_{j:j < i} x_{ij} + \sum_{j:j > i} x_{ji} = 2, \quad i \in V - \{0\} \\
 & \sum_{j:j \neq 0} x_{0j} = 2m \\
 & \sum_{\substack{i,j \in S \\ i < j}} x_{ij} \leq |S| - l(S), \quad S \subset V - \{0\}, |S| \geq 3 \\
 & x_{ij} \in \{0, 1\}, \quad (i, j) \in E
 \end{aligned}$$

4.2 FACILITY LOCATION AND NETWORK DESIGN MODELS

Another widely used class of operations research models focuses on the optimal location of “facilities” and how they are connected to the locations that they are expected to serve. Note that, by “facilities” we can mean anything from the physical location of warehouses and depots to the location of day-care centers and rain gauges. In addition, we are interested in the

optimal routes between these facilities and the end users of their services. In this section we will examine some common location and design models that occur in many operations research applications.

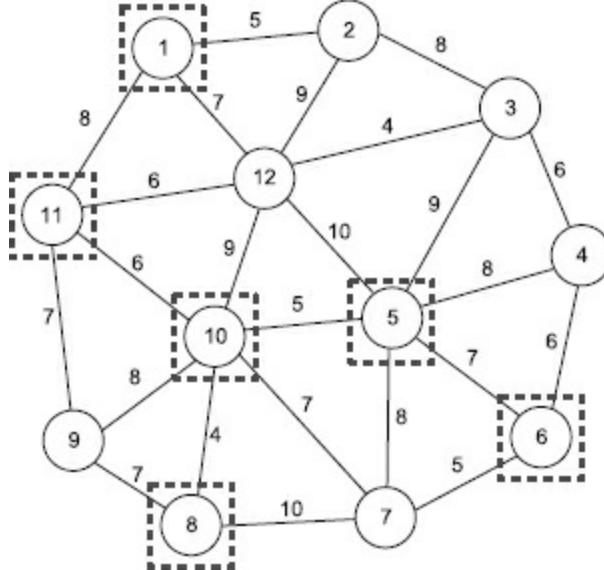
Each of these models is phrased in terms of graph theory. Consider a graph $G = (V, E)$, and assume that there are demand nodes $I \subseteq V$ that are known and fixed and a set $J \subseteq V$ of possible facility locations; note that I and J may have common nodes. The edges $(i, j) \in E$ indicate that it is possible for the demand at vertex i (j) to be directly satisfied if a facility is located at vertex j (i); the distance between nodes i and j is given by d_{ij} , where distance is a measure not just of physical distance but also of other factors such as time or cost.

Facility Location Models

Facility location models attempt to find the best location for new facilities on some network. As noted before, the potential locations for these facilities are fixed. This assumption to the problem makes the problem discrete in nature; if we allow the facilities to be anywhere in the plane, the problem becomes continuous. We focus only on the discrete problem in this section.

There are various facility location models in the literature, but most can be classified by their objective: maximum distance models and total or average distance models. A recent review by Current, Daskin, and Schilling [25] provides a good overview of these models.

FIGURE 4.1 Example problem used for facility location models.



Throughout this section we will use the example given in [Figure 4.1](#), where each of the $n = 12$ nodes $I = \{1, 2, \dots, 12\}$ corresponds to the customers and the nodes $J = \{1, 5, 6, 8, 10, 11\}$ in dashed boxes are the possible facility locations. The weight on each edge (i, j) corresponds to direct distances between i and j . Thus, if two nodes are not connected by an edge, their distance d_{ij} would equal the length of the shortest path in this graph between the two nodes; for example, the distance $d_{1,3}$ between nodes 1 and 3 equals 11, because its shortest path $1 \rightarrow 12 \rightarrow 3$ has length 11.

Regardless of the model, we need to identify which potential locations are actually going to house a facility. To this end, we naturally define the variables

$$(4.2) \quad x_j = \begin{cases} 1, & \text{if node } j \text{ is the location of a facility,} \\ 0, & \text{otherwise.} \end{cases}$$

for each $j \in J$. Each of the models presented in this section uses these variables, along with potentially other variables, in order to identify facility locations.

Maximum Distance Models In maximum distance models, we assume that the maximum distance between any two locations (demand nodes and potential facility locations) is known or can be easily calculated. Typically, in such models, the entire demand for a location is completely satisfied by the facility to which it is assigned.

In one class of models, we are given a maximum distance D at which a

demand node $i \in I$ can be satisfied by a facility $j \in J$. For such cases, we define the neighborhood of $i \in I$

$$N_i = \{j \in J : d_{ij} \leq D\}$$

as the set of all facilities j that can serve node i . In our first model, using variables x_j defined in (4.2), we want each demand node $i \in I$ to be associated with a facility; thus, at least one of the facilities in the neighborhood N_i must be selected. This leads to the covering constraint

$$\sum_{j \in N_i} x_j \geq 1.$$

If our goal is to minimize the number of facilities needed, we end up with the *Set Covering Location Problem*, given in Integer Program 4.2. Note that we can put weights on each variable in the objective to indicate preferences or costs on the potential facility locations.

Integer Program 4.2 Formulation of Set Covering Location Problem

$$\begin{aligned} \min \quad & \sum_{j \in J} x_j \\ \text{s.t.} \quad & \sum_{j \in N_i} x_j \geq 1, \quad i \in I \\ & x_j \in \{0, 1\}, \quad j \in J \end{aligned}$$

■ EXAMPLE 4.3

Let's solve the set covering location problem using the example given in [Figure 4.1](#). Calculating the distances between every customer i and potential facility location j generates the distance matrix

$$(4.3) \quad \mathbf{d} = \begin{bmatrix} 0 & 17 & 23 & 18 & 14 & 8 \\ 5 & 17 & 20 & 22 & 18 & 13 \\ 11 & 9 & 12 & 17 & 13 & 10 \\ 17 & 8 & 6 & 17 & 13 & 16 \\ 17 & 0 & 7 & 9 & 5 & 11 \\ 23 & 7 & 0 & 15 & 12 & 18 \\ 21 & 8 & 5 & 10 & 7 & 13 \\ 18 & 9 & 15 & 0 & 4 & 10 \\ 15 & 13 & 20 & 7 & 8 & 7 \\ 14 & 5 & 12 & 4 & 0 & 6 \\ 8 & 11 & 18 & 10 & 6 & 0 \\ 7 & 10 & 16 & 13 & 9 & 6 \end{bmatrix}.$$

If we set our maximum distance $D = 9$, our neighborhood sets N_i are

$$\begin{array}{ll} N_1 = \{1, 11\} & N_7 = \{5, 6, 10\} \\ N_2 = \{1\} & N_8 = \{5, 8, 10\} \\ N_3 = \{5\} & N_9 = \{8, 10, 11\} \\ N_4 = \{5, 6\} & N_{10} = \{5, 8, 10, 11\} \\ N_5 = \{5, 6, 8, 10\} & N_{11} = \{1, 10, 11\} \\ N_6 = \{5, 6\} & N_{12} = \{1, 10, 11\} \end{array}$$

Using these sets, we find that three locations are needed for this problem. One such set of locations is $\{1, 5, 8\}$.

Now suppose not every node $i \in I$ can have its demand satisfied. This may be due to budget restrictions or a similar limitation. Typically, this results in a limit on the number of facilities that can be opened. In this case, our objective will be to maximize the demand covered by the open facilities.

To model this, let h_i be the demand at node i and p denote the number of facilities we wish to locate. We need not only variables (4.2) to indicate which locations receive facilities but also variables to indicate which customers are to receive their demand; thus, for each $i \in I$ we let

$$y_i = \begin{cases} 1, & \text{if node } i \text{ has its demand satisfied by some facility,} \\ 0, & \text{otherwise.} \end{cases}$$

We can then extend the set covering problem (4.2) to form the *Maximal Covering Location Problem*, given in [Integer Program 4.3](#). Note that the objective function (and constraints) of [Integer Program 4.3](#) ensures that we cover as many demand sites as possible using only p facilities.

Integer Program 4.3 Formulation of Maximal Covering Location Problem

$$\begin{aligned}
\max \quad & \sum_{i \in I} h_i y_i \\
\text{s.t.} \quad & \sum_{j \in N_i} x_j \geq y_i, \quad i \in I \\
& \sum_{j \in J} x_j = p \\
& x_j \in \{0, 1\}, \quad j \in J \\
& y_i \in \{0, 1\}, \quad i \in I
\end{aligned}$$

■ EXAMPLE 4.4

Using the example from [Figure 4.1](#), we saw in Example 4.3 that every customer's demand can be met if we use at least three facilities. Suppose that we can only afford to build and maintain $p = 2$ such facilities, and that the demand values h_i for each customer are

$$(4.4) \quad \mathbf{h} = (100, 90, 110, 120, 80, 100, 95, 75, 110, 90, 120, 85).$$

Putting this information and the distance matrix \mathbf{d} given in (4.3) into Integer Program 4.3 indicates that using facilities at locations 5 and 11 allows us to cover 1085 units of demand (out of 1175 total), with only customer 2 not having its demand met.

A third model attempts to minimize the maximum distance between a demand site and its closest facility. Here, we want every customer's demand to be met, using only p facilities on our network. As in the previous models, let x_j indicate whether a facility is located at node j . However, unlike in the maximal covering location problem (4.3) where we were interested only in those customers whose demand is to be met, we are now also interested in determining which facility is used to meet that demand. To this end, for each $i \in I$ and $j \in J$ we define the variables

$$y_{ij} = \begin{cases} 1, & \text{if demand node } i \text{ has its demand satisfied by facility } j, \\ 0, & \text{otherwise.} \end{cases}$$

With these variables, we need a constraint to ensure that each customer's demand is met by exactly one facility, which can be written as

$$\sum_{j \in J} y_{ij} = 1, \quad i \in I.$$

In addition, we need to ensure that customer i 's demand is met by facility j only if facility j is open. To ensure this, we need the constraints

$$y_{ij} \leq x_j.$$

If we let W be the maximum distance between a demand node i and its assigned facility j , we have the traditional constraints

$$\sum_{j \in J} d_{ij} y_{ij} \leq W, \quad i \in I$$

associated with a minimax problem. Putting this all together, we can define the *p-center problem*, which is given in [Integer Program 4.4](#).

Integer Program 4.4 Formulation of p-Center Problem

$$\begin{aligned} & \min \quad W \\ & \text{s.t.} \\ & \quad \sum_{j \in J} y_{ij} = 1, \quad i \in I \\ & \quad \sum_{j \in J} x_j = p \\ & \quad y_{ij} \leq x_j, \quad i \in I, j \in J \\ & \quad \sum_{j \in J} d_{ij} y_{ij} \leq W, \quad i \in I \\ & \quad x_j \in \{0, 1\}, \quad j \in J \\ & \quad y_{ij} \in \{0, 1\}, \quad i \in I, j \in J \end{aligned}$$

We can easily alter this model to handle cases where we are interested in the maximum total demand covered by each facility or cases where the total number of facilities p is itself variable.

■ EXAMPLE 4.5

We again use the example from [Figure 4.1](#), as well as the demand values h_i given in (4.4) and the distance matrix \mathbf{d} given in (4.3). If we can have only $p = 2$ facilities, solving Integer Program 4.4 indicates that we can satisfy the demands from every customer using facility locations 5 and 11 if we allow a maximum distance of 13 between customers and facilities. The facility at location 5 would satisfy the demands of customers 3, 4, 6, 7, and 8, while location 11 services the rest.

Total or Average Distance Models Many other facility location models are interested in minimizing the total “distance” between facilities and their demand sites. This occurs not only in logistics, which are interested in the locations of plants and warehouses in relation to the location of its customers, but also in presidential election campaigns, which want to place local

campaign headquarters as close to as many people as possible. Let's explore some common approaches to these problems.

Our first problem is similar to the p -center problem discussed earlier. In that problem, we wanted to minimize the maximum distance between a facility and all its demand sites. In this new model, we are simply interested in the total distance; thus, constraint

$$\sum_{j \in J} d_{ij} y_{ij} \leq W, \quad i \in I$$

in the p -center problem gets transformed into the objective function directly. This new problem, referred to as the p -median problem, is given in Integer Program 4.5.

Integer Program 4.5 Formulation of p -Median Problem

$$\begin{aligned} \min \quad & \sum_{i \in I} \sum_{j \in J} h_i d_{ij} y_{ij} \\ \text{s.t.} \quad & \sum_{j \in J} y_{ij} = 1, \quad i \in I \\ & \sum_{j \in J} x_j = p \\ & y_{ij} \leq x_j, \quad i \in I, j \in J \\ & x_j \in \{0, 1\}, \quad j \in J \\ & y_{ij} \in \{0, 1\}, \quad i \in I, j \in J \end{aligned}$$

In this case, the objective is the weighted distance between demand node i and facility location j , where the weight is the total demand at node i .

■ EXAMPLE 4.6

If we take the data from Example 4.5 and use them to formulate the p -median problem, we find that the total distance (weighted by the customer demands) is minimized by having facilities at locations 6 and 11. Note that this is different from the p -center problem, which had locations 5 and 11. Here, location 6 satisfies the demands of customers 4, 5, 6 and 7, while location 11 manages the rest.

The p -median problem makes various assumptions that are hidden in the model. First, it assumes that each potential facility location j incurs the same setup cost. Second, it assumes that each facility location does not have an

upper bound on the amount of demand it can handle. The *fixed-charged location problem* attempts to relax each of these assumptions. To do this, suppose that f_j is the fixed-cost associated with locating a facility at site j and C_j is the capacity of a facility at site j . We add the fixed charge to our objective function, while the facility capacities are used in the constraint

$$\sum_{i \in I} h_i y_{ij} \leq C_j x_j,$$

which not only limits the production at site j but also replaces the constraints $y_{ij} \leq x_j$ restricting production to only open sites. In addition, we have a cost α per unit demand per unit distance associated to the weighted distance summation in the objective. Thus, the fixed-charged location problem can be formulated as in Integer Program 4.6.

Integer Program 4.6 Formulation of Fixed-Charge Location Problem

$$\begin{aligned} \min \quad & \sum_{j \in J} f_j x_j + \alpha \sum_{i \in I} \sum_{j \in J} h_i d_{ij} y_{ij} \\ \text{s.t.} \quad & \sum_{j \in J} y_{ij} = 1, \quad i \in I \\ & \sum_{j \in J} x_j = p \\ & \sum_{i \in I} h_i y_{ij} \leq C_j x_j, \quad j \in J \\ & x_j \in \{0, 1\}, \quad j \in J \end{aligned}$$

This model is very similar to those studied in Section 3.1. Note that we can further relax this model by eliminating the restriction on the number of facilities to be built and allowing any number of facilities we need.

■ EXAMPLE 4.7

If we take the data from Example 4.6, with $\alpha = 1$, and add the fixed costs

$$\mathbf{f} = (5000, 7000, 6000, 4000, 7000, 12000)$$

and the facility capacities

$$\mathbf{C} = (600, 650, 600, 500, 650, 800)$$

and solve the fixed-charge location problem, we find that the minimum combined cost and distance is obtained by having facilities at locations 1 and

6, where location 1 handles the customers 1, 2, 9, 10, 11, and 12, while location 6 handles the rest. This solution remains optimal, even if we allow the use of more facilities.

Network Design

You may have noticed that facility location problems centered on whether a given node is to open as a facility. What about problems determining whether an edge between two nodes is to be used? This problem is the basis of *Network Design Problems*, also known as *Fixed Charge Network Flow Problems*.

The basic formulation of a network design problem is a modification of a network flow model, first seen in Section 2.9. Recall that the linear program for this model was

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = b_i, \quad i \in V \\ & 0 \leq x_{ij} \leq u_{ij}, \quad (i, j) \in A, \end{aligned}$$

where $G = (V, A)$ was a directed network with costs c_{ij} associated with each arc $(i, j) \in A$. The variables x_{ij} indicated the amount of flow on arc (i, j) , while the parameters b_i indicated the balance of flow at node i —if $b_i > 0$, then i was a supply node, meaning more flow left i than entered; if $b_i < 0$, then i was a demand node. For the network design problem, we now associate a fixed one-time cost $f_{ij} > 0$ on each arc (i, j) , where we pay this cost only if we send any flow across (i, j) . We can model this situation similar to the fixed-charge cases we saw in Section 3.1 by letting

$$y_{ij} = \begin{cases} 1, & \text{if arc } (i, j) \text{ is to be used,} \\ 0, & \text{otherwise.} \end{cases}$$

and modifying the upper bound of x_{ij} to

$$x_{ij} \leq u_{ij} y_{ij}.$$

We can then add the fixed costs to our objective to obtain the network design problem shown in Integer Program 4.7.

Integer Program 4.7 Formulation of Network Design Problem

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

s.t.

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = b_i, \quad i \in V$$

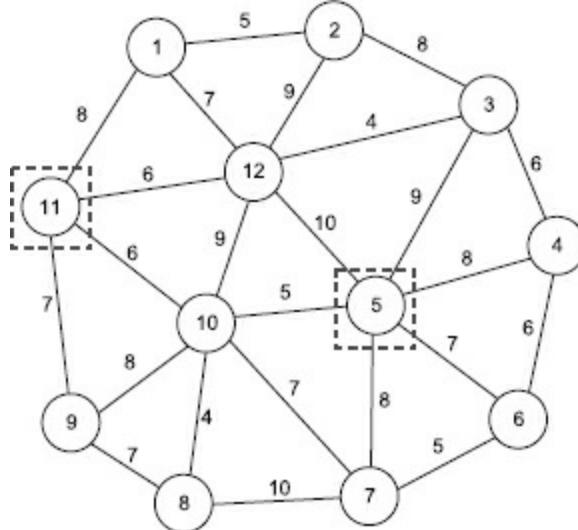
$$0 \leq x_{ij} \leq u_{ij}y_{ij}, \quad (i, j) \in A$$

$$y_{ij} \in \{0, 1\}, \quad (i, j) \in A$$

■ EXAMPLE 4.8

Let's consider the problem given in [Figure 4.2](#), where the distances d_{ij} between each pair of connected nodes are given and where the demands \mathbf{h} of each

[FIGURE 4.2](#) Graph used in Example 4.8 with distances d_{ij} .



customer are given in (4.4). Suppose we determine that we are going to operate facilities at locations 5 and 11, and we wish to satisfy customer demands from these facilities at minimum cost using only the edges of the graph. It is known that each edge (i, j) can handle at most $u_{ij} = u_{ji} = 175$ units of demand in each direction, and the cost to send one unit of demand from i to j along an edge is equal to $\$10d_{ij}$. Furthermore, if we are to use edge (i, j) to send demand from i to j , a one-time fixed cost f_{ij} is assessed, with the same fixed cost associated with sending demand from j to i (hence, both fixed costs

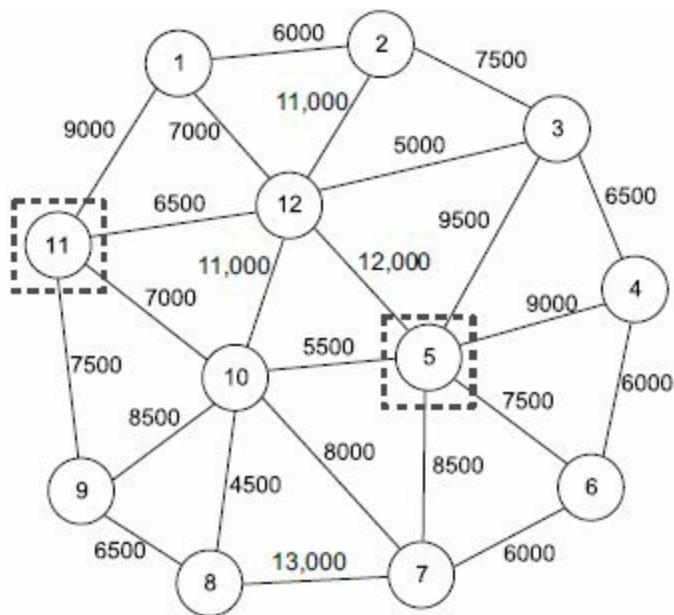
may be assessed); these fixed-charge values are shown in [Figure 4.3](#).

To model this as a network design problem, we will need one additional node s to act as the source and two additional arcs $(s, 5)$ and $(s, 11)$ each with cost 0 and no upper bound (see [Figure 4.4](#)). These arcs will be used to indicate which customers' demands are satisfied by these facilities. If we set $b(s) = 1175$ (total customer demands), and for every other node i we set $b(i) = -h(i)$, we get a minimum cost of \$158,150, with the resulting flows given in [Figure 4.5](#).

4.3 APPLICATIONS IN THE AIRLINE INDUSTRY

Operations research models and techniques have been employed by the airline industry since the 1950s; in fact, the Airline Group of the International Federation of Operational Research Societies (AGIFORS), a professional society concerning OR within the airline industry, has been active since 1961. These models have produced billions of dollars in savings by improving the operations of airlines in many areas.

FIGURE 4.3 Graph used in Example 4.8 with fixed costs f_{ij} .



Optimization models generally appear in schedule planning, in which

future airplane and crew schedules are determined so as to maximize the airline's profit. Because of the numerous issues involved with these decisions, such as the actual flights, different aircraft types, maintenance requirements, and crew work rules, to name a few, the industry typically decomposes the problem into various subproblems. These include

FIGURE 4.4 Graph of Example 4.8 with source node and arcs to facilities added.

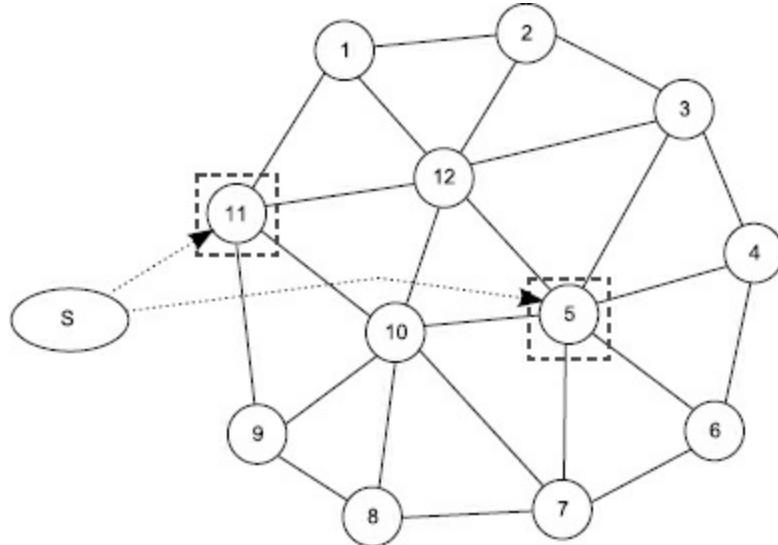
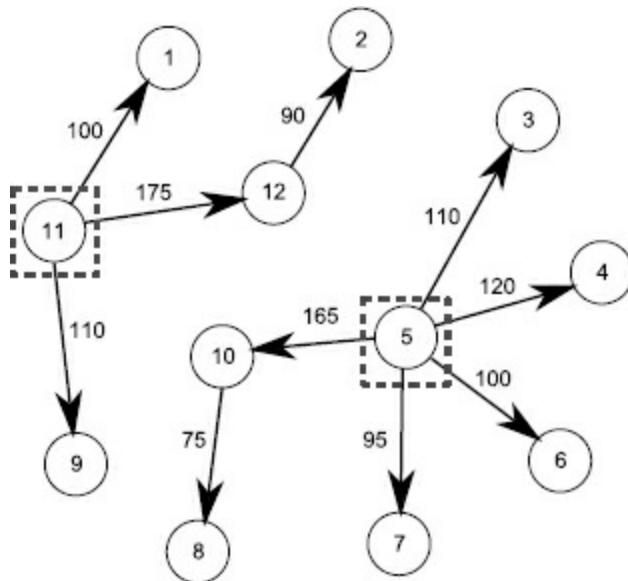


FIGURE 4.5 Solution to Example 4.8.



- *Schedule Design*, where the airline determines which airports to serve, how often, and how to schedule flights.

- *Fleet Assignment*, where each flight on the schedule is matched to a particular class of aircraft.
- *Aircraft Maintenance Routing*, where individual airplanes are assigned to flight segments.
- *Crew Scheduling*, where flight crews are determined and then paired to individual flights.

Each of these problems are computationally difficult to solve, given the large number of variables and constraints; for example, the crew scheduling problem often has billions of variables for large airlines [50]. Typically, heuristic methods are employed that attempt to quickly generate a feasible solution that is “close to” but not guaranteed to be optimal. However, much research is still being done to find optimal solutions to these problems.

In this section we will examine some of the classic problems that the airline industry faces: (1) fleet assignment, (2) aircraft routing, and (3) crew scheduling. For each of these problems we will provide a brief overview of the problem and discuss integer programming models for them. We then illustrate the first two models using an example situation. These problems are not the only ones analyzed by the airline industry, of course; the books by Bazargan [8] and Yu [91] introduce other problems solved using operations research techniques.

Throughout this section we will assume that the schedule design problem has been solved, and that there is a flight schedule. The following example provides some data with which we will explore our models.

■ EXAMPLE 4.9

Midwest Airline provides regional airline service to a small number of cities in the midwestern United States, as well as to some major cities in the eastern and southern United States. All its flights are based out of its hub in Indianapolis, Indiana, so that flights between other cities must travel through Indianapolis. Currently it provides airline service to the following cities: New York, Philadelphia, Atlanta, Detroit, Chicago, Minneapolis, and Dallas. [Table 4.1](#) gives its flight schedule for the next quarter. All times given are local—Chicago, Minneapolis, and Dallas are in the Central Time Zone , while the remaining cities (including Indianapolis) are in the Eastern Time Zone; recall that the Central Time Zone is 1 hour behind the Eastern Time Zone.

Fleet Assignment

The fleet assignment problem is to assign a particular aircraft type (fleet type) to each particular route on the flight schedule so as to minimize costs. By fleet type, we refer to the different classes of airplanes used by the airline; for example, an airline could use Boeing 737 and Boeing 747 as their two classes of aircraft. Typically, the costs decompose into (a) operating costs and (b) spill costs. Operating costs depend on the aircraft type, while spill costs represent the loss of revenue when there are more passengers wanting to fly a particular route than an aircraft can hold.

Our main goal is to assign flights to fleet types, which naturally leads to the decision variables

$$(4.5) \quad x_{f,k} = \begin{cases} 1, & \text{if flight } f \text{ is assigned to fleet type } k, \\ 0, & \text{otherwise.} \end{cases}$$

Note that these variables give the constraint

$$(4.6) \quad \sum_{k \in \text{Fleet}} x_{f,k} = 1,$$

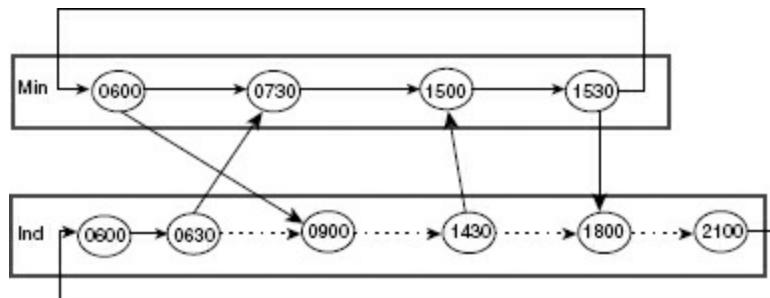
indicating that flight f must be assigned to exactly one fleet type. However, these variables do not give us everything we need. For example, we need to make sure that if we are to assign a flight to a specific fleet type, that type of aircraft is available for that flight at the airport of origin. Hence, we must also keep track of the number of aircraft of each fleet type at an airport during any time period.

To do this, we employ a network structure, where the nodes correspond to the times and locations of each flight departure and arrival, and the arcs correspond to each flight and to the time spent by an aircraft on the ground. Those aircraft staying overnight at an airport are indicated by an arc from the “latest” node to the “earliest” node associated with that location. Note that, for each node in the network, there are exactly two ground arcs incident to it —a “before” and an “after” arc. Because the nodes incorporate both location and time, it is often referred to as a *time-space network*.

TABLE 4.1 Flight Schedule for Midwest Airline Example

Flight No.	Origin	Departure Time	Destination	Arrival Time	Distance	98% Max
105	Min	0600	Ind	0900	502	84
110	Ind	0600	Chi	0600	177	90
153	Det	0600	Ind	0700	251	67
161	Ind	0600	NYC	0800	662	95
188	Ind	0600	Atl	0800	433	75
101	Ind	0630	Min	0730	502	62
121	Chi	0700	Ind	0900	177	93
194	Atl	0700	Ind	0900	433	63
145	Ind	0730	Det	0830	251	79
165	Ind	0730	NYC	0930	662	86
174	Ind	0730	Phl	0900	585	78
140	Dal	0800	Ind	1100	761	62
181	Phl	0830	Ind	1000	585	60
133	Ind	0900	Dal	1000	761	57
190	Ind	0900	Atl	1100	433	61
169	NYC	0900	Ind	1100	662	92
176	Ind	1100	Phl	1230	585	57
114	Ind	1200	Chi	1200	177	59
170	NYC	1230	Ind	1430	662	67
196	Atl	1300	Ind	1500	433	60
148	Ind	1330	Det	1430	251	54
183	Phl	1330	Ind	1500	585	55
129	Chi	1400	Ind	1600	177	60
157	Det	1400	Ind	1500	251	61
102	Ind	1430	Min	1530	502	58
107	Min	1500	Ind	1800	502	68
142	Dal	1500	Ind	1800	761	59
179	Ind	1500	Phl	1630	585	45
167	Ind	1500	NYC	1700	772	68
191	Ind	1500	Atl	1700	433	58
172	NYC	1600	Ind	1800	662	88
197	Atl	1600	Ind	1800	433	57
136	Ind	1630	Dal	1730	761	66
185	Phl	1630	Ind	1800	585	60
131	Chi	1800	Ind	2000	177	64
151	Ind	1800	Det	1900	251	59
118	Ind	1930	Chi	1930	177	88
159	Det	2000	Ind	2100	251	90

FIGURE 4.6 Example network for fleet assignment problem.



■ EXAMPLE 4.10

In [Figure 4.6](#) we show part of the network for Minneapolis and

Indianapolis. The cross-arcs indicate flights between the two cities, while the ground arcs are contained inside each airport's "box." The nodes correspond to times that a flight either originated or ends at that location; for Indianapolis, there are additional nodes that are not included in the graph (due to lack of space).

Using this network, we can now keep track of aircraft types between flights. If we define the variables

$$g_{e,k} = \text{number of aircraft of fleet type } k \text{ on the ground during the time interval described by edge } e,$$

we can ensure that an aircraft of each fleet type k is available at an airport at the proper time. Consider node j , and let j^- and j^+ be the nodes in the network so that (j^-, j) and (j, j^+) are ground arcs. Variable $g_{(i, j), k}$ gives the number of aircraft of fleet type k remaining on the ground between time periods i and j . To maintain aircraft balance at node j , we have the constraint

$$(4.7) \quad g_{(j^-, j), k} + \sum_{f: \text{flight arrives at } j} x_{f, k} - \sum_{f: \text{flight leaves from } j} x_{f, k} = g_{(j, j^+), k},$$

where the left-hand side indicates the change in the number of aircraft at the time associated with node j .

■ EXAMPLE 4.11

Consider the node 0730 associated with Minneapolis in [Figure 4.6](#). Since there is only one arriving flight into this node, its balance constraint (4.7) for fleet type k is

$$g_{(Min0600, Min0730), k} + x_{(Ind0630, Min0730), k} = g_{(Min0730, Min1500), k}.$$

If we look at node 1530 for Minneapolis, because of the flight departing Minneapolis at this time, we get the balance constraint

$$\underline{g_{(Min1500, Min1530), k} - x_{(Min1530, Ind1800), k} = g_{(Min1530, Min0600), k}}.$$

There are additional constraints we could add to this model. For example, if there are a fixed number of each fleet type available to the airline, this needs to be a set of constraints. We can calculate the total numbers of aircraft (for each fleet type) used in the problem by adding the ground arc variables corresponding to the overnight stays (such as variable $g_{(Min1530, Min0600), k}$ in our example). In addition, if there are flights that typically have large demands, we can restrict which fleet types can be used for those flights.

■ EXAMPLE 4.12

Continuing on with our example, Midwest Airlines is interested in determining how many planes to procure in order to meet their proposed flight schedule given in [Table 4.1](#). They are interested in using two types of aircraft: Embraer ERJ-170 and ERJ-190. The ERJ-170 can hold 70 passengers, while the ERJ-190 can hold 98; thus, they cannot use the ERJ-170 for every flight, given their expected demands. Our network has 38 flights and 55 ground arcs, which generates an integer program (using 2 fleet types) with 184 variables and 146 constraints.

To estimate costs, they consider only operating costs, which are calculated by using an estimate of the cost per available seat mile (CASM) multiplied by the total number of seats and total mileage for each flight. For example, flight 170 from New York to Indianapolis travels 662 miles. If the CASM for an ERJ-170 is calculated at \$0.03, then the operating cost for using an ERJ-170 for this flight would be $\$0.03 \times 662 \times 70 = \1390.20 . Using a CASM for an ERJ-190 of \$0.04, we get an optimal daily cost of \$49,955.40 using eight ERJ-170 aircraft and nine EJR-190 aircraft. The assignment of flights to fleet types is given in [Table 4.2](#). In addition, the overnight stays for each aircraft type are given belows.

Airport	ERJ-170	ERJ-190
Min	1	1
Chi	0	1
Dal	1	0
Det	2	0
NYC	1	0
Phl	1	0
Atl	1	0
Ind	1	7

TABLE 4.2 Flight Schedule for Midwest Airline Example

Flight No.	Origin	Departure Time	Destination	Arrival Time	Fleet Type
105	Min	0600	Ind	0900	ERJ-190
110	Ind	0600	Chi	0600	ERJ-190
153	Det	0600	Ind	0700	ERJ-170
161	Ind	0600	NYC	0800	ERJ-190
188	Ind	0600	Atl	0800	ERJ-190
101	Ind	0630	Min	0730	ERJ-190
121	Chi	0700	Ind	0900	ERJ-190
194	Atl	0700	Ind	0900	ERJ-170
145	Ind	0730	Det	0830	ERJ-190
165	Ind	0730	NYC	0930	ERJ-190
174	Ind	0730	Phl	0900	ERJ-190
140	Dal	0800	Ind	1100	ERJ-170
181	Phl	0830	Ind	1000	ERJ-170
133	Ind	0900	Dal	1000	ERJ-170
190	Ind	0900	Atl	1100	ERJ-170
169	NYC	0900	Ind	1100	ERJ-190
176	Ind	1100	Phl	1230	ERJ-170
114	Ind	1200	Chi	1200	ERJ-170
170	NYC	1230	Ind	1430	ERJ-170
196	Atl	1300	Ind	1500	ERJ-170
148	Ind	1330	Det	1430	ERJ-170
183	Phl	1330	Ind	1500	ERJ-170
129	Chi	1400	Ind	1600	ERJ-170
157	Det	1400	Ind	1500	ERJ-170
102	Ind	1430	Min	1530	ERJ-170
107	Min	1500	Ind	1800	ERJ-170
142	Dal	1500	Ind	1800	ERJ-170
179	Ind	1500	Phl	1630	ERJ-170
167	Ind	1500	NYC	1700	ERJ-170
191	Ind	1500	Atl	1700	ERJ-170
172	NYC	1600	Ind	1800	ERJ-190
197	Atl	1600	Ind	1800	ERJ-190
136	Ind	1630	Dal	1730	ERJ-170
185	Phl	1630	Ind	1800	ERJ-190
131	Chi	1800	Ind	2000	ERJ-190
151	Ind	1800	Det	1900	ERJ-170
118	Ind	1930	Chi	1930	ERJ-190
159	Det	2000	Ind	2100	ERJ-190

It should be pointed out that this model does require a few assumptions. First, the model assumes that flight schedules repeat daily; however, many airlines operated under different schedules during the weekends. Second, the model assumes that flight demands are known and do not vary. Third, it is assumed that flying and ground times are constant, which is not often the case. However, even with these assumptions this basic model is used successfully by airlines to determine their fleet assignments. For instance, airlines that used fleet assignment models during the 1990s reported savings in the millions of dollars [1, 78, 89].

Aircraft Maintenance Routing

Once a fleet assignment has been determined, the next problem for the airline is to assign individual aircraft to flights. This problem, known as the Aircraft Maintenance Routing Problem, is to determine a sequence of flight legs for each aircraft, known as a routing or rotation, where the destination of one flight leg is the same as the origin of the next leg and the total operating cost is minimized (or revenue is maximized). Such a routing must satisfy the following conditions:

- Each flight leg must be covered by exactly one aircraft.
- The utilization loads of each aircraft must be balanced.
- Each sequence must include an overnight stay at an airport that contains a maintenance station, and this stay must occur at regularly specified intervals.

Typically, an airline does not require that this sequence start and end at the same airport or be of a specified time length. However, simplified versions of this problem often require these conditions. This problem is solved for each fleet type used by the airline, since our fleet assignment problem partitioned the flights by fleet type used.

Typically, airlines perform standard maintenance inspections on an aircraft every 3–4 days. These inspections, known as A-checks, can last anywhere from 3–10 hours, hence requiring an overnight stay to ensure that the aircraft does not miss any assigned flights. Other inspections, known as B-, C-, and D-checks, are more involved and can involve the removal of an aircraft from the schedule for an extended period; these typically only occur either every few months, for B-checks, or every few years, for C- and D-checks. The aircraft maintenance routing problem deals only with A-checks.

One way to model this routing problem is by a *set partitioning model* with the addition of side constraints. We first saw the set partitioning problem in Section 3.2, where instead of using covering constraints

$$\sum_{j \in N_i} x_j \geq 1$$

in the set covering problem we use partitioning constraints

$$\sum_{j \in N_i} x_j = 1.$$

Each variable x_j will correspond to a valid routing of an aircraft, where a routing is valid if it includes an overnight stay at a maintenance airport location every 3–4 days; this can be ensured by using only routes that begin and end at the same airport, has length of 3 or 4 days, and includes such an overnight stay. These valid routings are typically enumerated using some graph search algorithm, where nodes correspond to flight origins or destination locations and times, and arcs correspond to either actual flights (origin to destination) or from a destination of one flight to the origin of another flight (in the same city) to which that fleet type can also be assigned. Once these routes are determined we then define

$$x_j = \begin{cases} 1, & \text{if routing } j \in R_k \text{ is used,} \\ 0, & \text{otherwise,} \end{cases}$$

where R_k is the set of valid routes for aircraft in fleet type k . Of course, using these as our variables typically results in very large problem sizes, where the number of variables can easily reach a few million.

■ EXAMPLE 4.13

Consider the fleet assignment for Midwest Airlines given in [Table 4.2](#). If we assume that an aircraft requires at least 30 minutes from its arrival at an airport before it can depart (to prepare the aircraft for the next flight), there are 36 possible 1-day routes to which an ERJ-190 aircraft can be assigned and 107 1-day routes for an ERJ-170 aircraft. Some of these routes consist of only one flight, while others consist of as many as four flights; for example, an ERJ-170 aircraft can be assigned to flights 140, 114, 129, and 151, resulting in flights from DAL at 0800 to IND at 1100, from IND at 1200 to CHI at 1200, from CHI at 1400 to IND at 1600, and from IND at 1800 to Det at 1900.

If we form 3-day routes that include at least one overnight stay at IND for maintenance, we find 3217 such routings for ERJ-170 aircraft and 1178 routings for ERJ-190 aircraft.

There are typically two types of constraints in this model. The first, called the *flight cover constraints*, ensure that a flight f is covered by a chosen route. If, for flight f and day $d = 1, 2, 3$, we let

$$N_f^d = \{j : \text{route } j \text{ includes flight } f \in F_k \text{ on day } d\},$$

where F_k denotes the set of flights to be assigned to an aircraft in fleet type k , our constraint for flight f and day d would then be

$$\sum_{j \in N_f^d} x_j = 1.$$

The second class of constraints involves the number of available aircraft. Here, if we let n_k denote the number of available aircraft of fleet type k , we must ensure that

$$\sum_{j \in R_k} x_j \leq n_k.$$

For our objective function, we have multiple choices. We could simply minimize some cost function, where we assign costs to routes that reflect a penalty for bad routes. Such penalties could occur if a route has bad connection times or either small or large utilizations. Another choice would be to maximize the opportunities for maintenance. Thus, those routes with multiple times for maintenance would be preferable.

Our full model could then be formulated for each fleet type k as

$$\begin{aligned} \min / \max \quad & \sum_{j \in R_k} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in N_f^d} x_j = 1, \quad f \in F_k, \quad d \in \{1, 2, 3\} \\ & \sum_{j \in R_k} x_j \leq n_k \\ (4.8) \quad & x_j \in \{0, 1\}, \quad j \in R_k. \end{aligned}$$

■ EXAMPLE 4.14

For the Midwest Airline example, we will try to maximize the number of overnight stays at IND during a 3-day period. Solving (4.8) for the ERJ-190 flights given in [Table 4.2](#) yields a total of 21 overnight stays using a fleet size of 9. One possible set of routes is as follows:

Routes	Day 1	Day 2	Day 3	Maintenance Ops
1	188–197	174–185–118	121	2
2	161–169	145–159	161–169	3
3	110–131	161–169	188–197	3
4	101	105	174–185	2
5	174–185	165–172	165–172	3
6	165–172	188–197	145–159	3
7	145–159	110–121	110–131	3
8	121	101	105–118	1
9	105–118	131	101	1

If we attempt to solve (4.8) for the ERJ-170 flights, we find that there is no feasible solution! In fact, we find that we need at least 11 aircraft to cover these flights. Why did our solution to Example 4.12 indicate only 8 aircraft are needed?

Unfortunately, it is not uncommon for there to be no feasible solution to this model. This could be due to the fact that, when we generated a fleet assignment, valid routings of individual aircraft were not taken into account. Also, requiring an overnight stay at a maintenance location can cause a problem, even if we extend the notion of a valid routing to include those that do not necessarily begin and end at the same location. Typically, a feasible solution can be found if we remove the fleet size restriction and allow more aircraft in the model.

Another typical cause of infeasibility is that flights in a schedule are not synchronized to ensure that one aircraft can fly multiple routes. For example, in the proposed schedule for Midwest Airline given in [Table 4.1](#), if we were to require at least 45 minutes after arrival for an aircraft to fly again, due to baggage and passenger unloading and loading, the aircraft assigned to flight 170, which arrives in Indianapolis at 1430, cannot be assigned to any of the flights 167, 179, or 191 that leave Indianapolis at 1500. One possible solution that frequently occurs is to alter the flight schedule to better synchronize some flights. For example, if we move flight 167 from a departure time of 1500 to 1530 out of Indianapolis, we can now assign the same aircraft to both of these flights. This situation commonly occurs, and is part of the overall modeling process we saw in Chapter 1.

■ EXAMPLE 4.15

Solving the maintenance problem for the ERJ-170 aircraft using 11 aircraft results in the same cost as in Example 4.12, since we did not consider the cost

of purchasing the aircraft in that problem. This is not an uncommon occurrence; in fact, the airline industry often faces this dilemma. One possible solution is to adjust the flight schedules to better accommodate these additional constraints. In our case, if we allow the use of 11 aircraft, and allow only 30 minutes between possible arrival and departure times for flights on the same route, we find that there are a total of 12 overnight stays at IND from the following routes:

Routes	Day 1	Day 2	Day 3	Maintenance Ops
1	190–196–136	140–114–129	133–142	2
2	148	157–151	153–190–196	1
3	179	181–102	107	1
4	167	170–191	194–114–129	1
5	191–170	194–176–183	167	1
6	181–176–183–151	153–133–142	179	1
7	194–114–129	167	170–191	1
8	153–133–142	179	181–143	1
9	157	148	157–151	1
10	107	190–196–136	140–102	1
11	140–102	107	176–183–136	1

Crew Scheduling

A problem that is often solved simultaneously with the aircraft routing problem is the Crew Scheduling Problem, which attempts to assign flight crews (pilots and cabin crews) to flights. This problem takes into account not only costs associated with a crew (including salary, benefits, and per diem expenses) but also the numerous complex rules designed by the Federal Aviation Administration (FAA), which ensure that the crews receive proper rest after a certain number of flight hours. For example, the 8-in-24 rule states that no more than 8 hours of flying may be assigned to a work crew during any 24-hour period unless a rest period of twice the flying time is given and at least 14 hours of rest is given after a period in which the 8-hour limit is exceeded.

Crew costs are typically the second largest annual expenditure an airline faces, with only fuel costs being larger. In fact, because crew costs are so large (easily exceeding \$1 billion annually for the major North American airlines) and are the biggest cost that an airline can control, much research has been done on this problem. However, given the size of crews, the number of potential crew members, and the number of flights, there can easily be

several billion crew combinations for the larger airlines. Thus, the crew scheduling problem is often very difficult to solve.

In fact, the crew scheduling problem is often decomposed into two sequentially solved subproblems—the *Crew Pairing Problem* and the *Crew Assignment Problem*. The crew pairing problem generates multiple-day work schedules (pairings) at minimum cost. Such pairings must satisfy various work rules and typically consist of a sequence of flights, within the same fleet type, that start and end at the crew’s home base city, which is where they actually live. These bases need not be the central hubs of an airline; for example, in the Midwest Airline example, a crew could have their base in Chicago or Atlanta and not necessarily in Indianapolis. The crew assignment problem combines the pairings generated by the pairing problem into month-long crew schedules, called *bidlines* or *rosters*, and assigns them to individual crew members. Since there are often multiple crew members living at the same crew base, this problem is not automatically solved once the pairings are generated.

Similar to the aircraft routing problem, we can model both the crew pairing problem and the crew assignment problem using a set partitioning model. For example, we can model the crew pairing problem by first defining the variables

$$x_j = \begin{cases} 1, & \text{if crew pairing } j \text{ is used} \\ 0, & \text{otherwise} \end{cases}$$

for each crew pair j and the partitioning constraints are given by

$$\sum_{j:f \in P_j} x_j = 1$$

for each flight $f \in F$, where P_j is the set of flights covered by crew pairing j . We typically also add “crew base constraints,” which bound the number of crew pairings based at a particular city; these constraints are typically of the form

$$l_k \leq \sum_{j \in PB_k} x_j \leq u_k,$$

where PB_k is the set of pairings based at city k , and l_k and u_k are the minimum and maximum number of crews we can base at city k , respectively. Our objective is typically to minimize costs, so that our crew pairing model can be formulated as

$$\begin{aligned}
\min \quad & \sum_{j \in P} c_j x_j \\
\text{s.t.} \quad & \sum_{j: f \in P_j} x_j = 1, \quad f \in F \\
& l_k \leq \sum_{j \in P_B k} x_j \leq u_k, \quad k \in B \\
(4.9) \quad & x_j \in \{0, 1\}, \quad j \in P,
\end{aligned}$$

where P is the set of all pairings and B is the set of all base locations.

Summary

In this chapter we explored a variety of OR models that are commonly used in business and industry. Each illustrates the usefulness of the optimization models and modeling techniques we saw in Chapters 2 and 3, and hopefully they motivate us to determine how to solve such problems. Of course, these models are just a few of the many classes of models and problems solved by OR techniques today. More applications and their solution approaches can be found by exploring the research literature.

EXERCISES

4.1 A local beverage supplier has a depot at location 1 from where they must deliver vending supplies to customers at locations 2, . . . , 7. Each customer has a demand for d_j units, and the distance between locations i and j is given by c_{ij} ; both sets of parameters are given in the table below. If each truck can carry at most 120 units, develop an integer program to minimize the total distance traveled so as to meet all demands.

		Distances (miles)							Deliver	
		Location								
Location	1	2	3	4	5	6	7			
1	—	30	20	10	10	20	15			
2	30	—	10	20	40	50	35	50		
3	20	10	—	30	10	40	20	75		
4	10	20	30	—	15	25	20	80		
5	10	40	10	15	—	30	25	40		
6	20	50	40	25	30	—	10	30		
7	15	35	20	20	25	10	—	60		

4.2 In Problem 4.1, now suppose that each truck can travel a distance of no more than 75 miles. Modify your model to include this constraint.

4.3 Meals on Wheels is a national nonprofit organization that provides lunch for people who cannot leave their homes during the day. In a small community, 12 meals need to be delivered over the next 2 hours using at most 4 cars. Each car can carry at most 4 meals. The meal setup time at each home varies and is given below.

	1	2	3	4	5	6	7	8	9	10	11	12
Setup time	10	12	8	7	15	12	10	11	9	12	8	15

In addition, the (symmetric) travel times between each of the 12 homes and main office (location 0) are given in the following table.

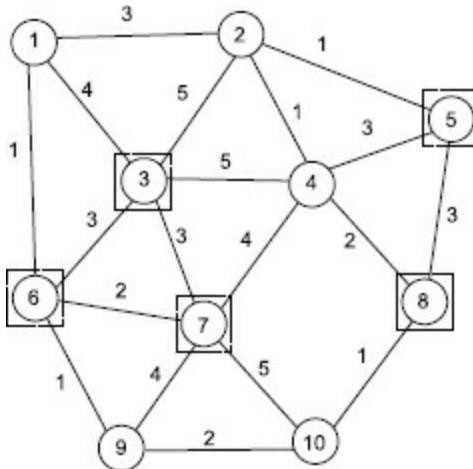
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	—	8	4	10	12	9	15	8	11	5	9	4	10
1	—	7	6	8	6	7	10	12	9	8	7	5	—
2	—	7	9	5	8	5	4	8	6	10	8	—	—
3	—	6	11	5	9	8	12	11	6	9	—	—	—
4	—	7	9	6	9	8	4	11	6	11	10	—	—
5	—	10	4	3	10	6	5	7	6	5	7	—	—
6	—	10	9	8	5	9	7	10	6	9	10	—	—
7	—	11	5	9	6	7	8	7	6	7	8	—	—
8	—	9	11	11	11	11	11	11	11	11	11	6	—
9	—	6	7	5	6	7	5	6	7	5	6	7	—
10	—	10	7	5	10	7	5	6	7	5	6	7	—
11	—	9	7	5	7	9	5	6	7	5	6	7	—

If all cars must return to the main office after completing their deliveries, how should the meals be delivered if we are interested in minimizing the total transportation time; the time required to return to the main office after the last delivery does not need to be counted.

Exercises 4.4–4.11 use the graph in [Figure 4.7](#), where the nodes $J = \{3, 5, 6, 7, 8\}$ in dashed boxes are possible facility locations, the nodes $I = \{1, 2, \dots, 10\}$ are customers, and the edge weights correspond to direct distances between nodes i and j . The total distance d_{ij} corresponds to the length of the shortest path between nodes i and j .

4.4 Suppose the allowable maximum distance between a customer and the facility it utilizes is $D = 4$. What is the fewest number of facilities needed to satisfy every customer's demand?

FIGURE 4.7 Graph for Exercises 4.4–4.11.



4.5 Continuing with Exercise 4.4, suppose that we can afford to use only $p = 1$ facility. If the customer demands are

$$(4.10) \mathbf{h} = (80, 100, 120, 75, 90, 110, 100, 85, 90, 105),$$

what is the maximum amount of customer demand that we can meet? Where is the facility located and which customer demands are unsatisfied?

4.6 In Exercise 4.4, what is the smallest maximum distance needed to ensure that every customer's demand can be satisfied by one facility? Where is that facility located?

4.7 Suppose we are interested in minimizing the total weighted distances between each customer and a single facility, where the weights are the customer demands. Using the demand vector \mathbf{h} given in (4.10), where should the facility be located and what is the minimum weighted distance?

4.8 In Exercise 4.7, suppose that a customer's demand can be satisfied by multiple facilities. Where should the facilities be located and what is the minimum weighted distance?

4.9 Suppose that there is a one-time fixed charge f_j associated with opening a facility at location $j \in J$ and a capacity \mathbf{C} associated with each facility. Using the demands in (4.10) and the fixed charges and facility capacities

$$\mathbf{f} = (f_3, f_5, f_6, f_7, f_8) = (4000, 3000, 5000, 2000, 3500)$$

$$\mathbf{C} = (C_3, C_5, C_6, C_7, C_8) = (650, 450, 800, 350, 500),$$

which facilities should be opened if we want to solve the fixed-charge location problem using two facilities? Assume that a customer's demand must be met by exactly one facility, and that $\alpha = \$1$.

4.10 In Exercise 4.9, suppose that a customer's demand can be satisfied by multiple facilities. Where should the facilities be located now in order to minimize total costs?

4.11 Suppose we have determined that facilities 5 and 6 are to open, and that we wish to satisfy all customer demands from these locations. If each edge (i, j) of G can handle at most 150 units of demand in each direction, and the cost associated with sending a unit of demand from i to j along (i, j) is $\$15d_{ij}$ in addition to the one-time fixed cost $\$(500 + 30d_{ij})$ for sending any demand from i to j , determine how we can satisfy all customer demands at minimum cost. Use the customer demands \mathbf{h} in (4.10), and that $\alpha = \$1$.

4.12 Consider the network design problem examined in Example 4.8. Suppose that, instead of automatically using facilities at locations 5 and 11, we want to determine which facilities among the set $J = \{1, 5, 6, 8, 10, 11\}$ to open. As in the fixed-charge location problem, we associate a fixed-charge cost \mathbf{f} and a capacity C_j with each facility opened. Using the data given in Example 4.7 for \mathbf{f} and each C_j , determine which facilities to open and which edges to use in order to satisfy all customer demands at minimum cost. Use $\alpha = \$1$.

4.13 (Based on Bloemhof-Ruwaard et al. [18]) In some cases, the waste generated by the production of material at a facility must be disposed of at special waste disposal locations. We need to identify which waste facilities to open, given a capacity for each waste unit and per-unit transportation costs for shipping waste from a plant facility to a waste facility. We can model this problem by extending our facility location models. Suppose we have the network given in [Figure 4.7](#) and let $W = \{4, 9, 10\}$ be possible location for our waste disposal facilities. At each plant $j \in J$, the amount of waste material produced is proportional (with proportionality constant $0 < \alpha_j < 1$) to the amount of goods produced; for example, if plant 5 produces 500 units, then it also produces $500\alpha_5$ units of waste. These proportionality constants are

$$(\alpha_3, \alpha_5, \alpha_6, \alpha_7, \alpha_8) = (0.1, 0.15, 0.2, 0.125, 0.175).$$

The one-time fixed cost associated with opening a waste disposal facility at $w \in W$ is

$$\mathbf{f}^W = (f_4^W, f_9^W, f_{10}^W) = (5000, 7000, 6000)$$

and their capacities are $(C_4^W, C_9^W, C_{10}^W) = (100, 140, 120)$. The cost of shipping one unit of waste from plant $j \in J$ to waste disposal unit $w \in W$ is $\$10d_{jw}$, where d_{jw} is the shortest distance from j to w , while the cost of shipping one unit of demand from plant j to customer k is $\$20d_{jk}$. Assuming that a customer can receive its demand from multiple plants and that the waste from a plant can be shipped to multiple disposal locations, determine how to satisfy customer demands and waste removal requirements at minimum cost. Where are the plants and waste disposal units located?

4.14 (Based on Sherali et al. [81]) Telecommunication companies are often concerned with the design of local access and transport area (LATA) networks that provide special access service through a local exchange facility. Within each LATA, a point of presence (POP) node is used to connect a local exchange to the “global” network. Each client in the local exchange can be connected directly to the POP if their requirements are large or they are connected with other clients through a hub node, where the transmissions are consolidated before being transmitted to the POP. This aggregation of client demands is done to save money, since direct connections to the POP node are very expensive.

Suppose we have 8 possible hub locations that, along with the POP node, are to be connected to 10 customers. Each hub used can handle a total demand of 200 units. The demand of each customer is given below.

	Customers									
	1	2	3	4	5	6	7	8	9	10
Demand	50	75	80	40	35	65	70	45	55	60

The per-unit cost of sending customer i 's demand to either node j or the POP, as well as the fixed-charge cost of using a hub, is given in [Table 4.3](#).

Finally, we require that between three and six hubs be used; the additional cost associated with opening the hubs depends on their quantity and is not linear, as seen in the following table.

TABLE 4.3 Transmission Costs for Exercise 4.14

	Hubs								
	1	2	3	4	5	6	7	8	POP
Cust. 1	20	35	25	40	15	30	45	35	150
Cust. 2	30	20	50	30	45	20	35	50	200
Cust. 3	45	50	30	55	35	25	40	40	175
Cust. 4	35	25	30	20	50	35	20	45	200
Cust. 5	40	50	30	40	45	25	50	25	250
Cust. 6	50	35	25	40	50	40	30	30	225
Cust. 7	35	25	40	25	20	50	25	25	150
Cust. 8	25	40	40	35	50	35	40	50	275
Cust. 9	40	20	35	50	35	45	30	45	225
Cust. 10	35	40	20	30	40	45	35	40	175
Fixed charge	5000	3500	6000	4000	5500	4500	4000	3500	

	Number of Hubs Used			
	3	4	5	6
Cost (\$)	40,000	60,000	70,000	75,000

Formulate and solve an integer program that determines which hubs to open and allocates customer demands to these hubs and the POP to minimize total cost.

4.15 Norris Airlines is a small regional airline servicing Chicago, IL, Minneapolis, MN, and Kansas City, MO. They are in the middle of revising their schedule and are interested in knowing the minimum number of aircraft needed to handle their proposed flights. All flights can be handled by the same type of aircraft, and an aircraft needs at least 30 minutes between flights for preparation. [Table 4.4](#) contains the proposed flight schedule, with all times given in military time, and the expected passenger amount. What is the minimum number of aircraft needed to fly each of their flights?

4.16 In Exercise 4.15, Norris is not convinced that using all 20 proposed flights is profitable, so it is willing to use only those that yield the greatest total profit.

TABLE 4.4 Flight Information for Norris Airlines (Exercises 4.15 and 4.16)

Origin	Departure Time	Destination	Arrival Time	Expected Passengers
Chi	0600	KC	0730	60
Min	0630	Chi	0800	45
Chi	0700	KC	0830	55
Min	0700	KC	0830	60
KC	0700	Chi	0830	55
Chi	0730	Min	0900	50
KC	0730	Min	0900	60
Chi	0900	Min	1030	50
KC	0900	Min	1030	40
Min	1000	KC	1130	45
Min	1100	Chi	1230	55
KC	1130	Chi	1300	50
KC	1300	Min	1430	40
Min	1400	KC	1530	50
Min	1600	Chi	1730	55
Chi	1630	KC	1800	70
KC	1700	Chi	1830	55
Chi	1700	Min	1830	50
KC	1730	Min	1900	65
Min	1800	KC	1930	60

Suppose that each flight from Chicago to Kansas City nets a profit of \$60 per passenger, from Chicago to Minneapolis nets \$65 per passenger, from Kansas City to Chicago nets \$85 per passenger, from Kansas City to Minneapolis nets \$75 per passenger, from Minneapolis to Chicago nets \$70 per passenger, and from Minneapolis to Kansas City nets \$85 per passenger. If each aircraft costs \$10,000 per day to operate (no matter the number of flights it flies), which flights should be kept and how many aircraft used in order to maximize its total daily profit?

4.17 How can we alter the VRP formulation (4.1) so that a route may contain only one customer?

4.18 How can we alter the VRP formulation (4.1) to the asymmetric case, where arc (i, j) may have a different cost from arc (j, i) ?

4.19 Generalize the network design model to include multiple commodity classes K , similar to the way we formulated the multicommodity flow problem in Section 2.9.

CHAPTER 5

INTRODUCTION TO ALGORITHM DESIGN

When people discuss deterministic operations research, they often refer to one of three aspects: modeling, algorithms to “solve” the model, and an analysis of both the algorithms used and the solution itself. Having been introduced to various optimization models, we now focus on deriving algorithms. While model building is perceived as creative, many say that the construction of algorithms is even more so. What is amazing is that, no matter whether the problem involves continuous or discrete variables, the basic approaches to optimization are the same.

In this chapter, we examine basic approaches to algorithm design by exploring approaches to solve discrete optimization problems. In particular, we explore algorithms that are time-efficient but that do not always find the optimal solution. Such heuristic methods are often used to solve hard discrete optimization problems due to computational difficulties; we saw examples of this in Chapter 3 for the traveling salesperson problem and sports scheduling. In Chapter 6, we will look at similar approaches to “solving” continuous optimization problems.

5.1 EXACT AND HEURISTIC ALGORITHMS

Our discussion of algorithm design begins with determining its overall goal. There are two general classes of optimization algorithms: **exact methods** that find a global optimal solution to the problem, no matter how long it takes, and **heuristic methods** that attempt to find a near-optimal solution quickly.

Exact methods use mathematical results to guarantee their solutions are optimal, but either they can solve problems efficiently, compared to the “size” of the problem, or they can require large amounts of time. Heuristics are typically algorithms that are based upon rules of thumb, common sense, or refinements of exact methods.

But why use heuristics at all? For many problems, and in particular most discrete problems, exact methods can be time consuming, and those who have need to solve real-world problems typically do not have the time required to guarantee optimality when a reasonable solution will suffice. In fact, an exact method might have to examine every feasible solution before confirming optimality. In these cases, heuristic methods that find good solutions quickly are often used.

For example, consider the traveling salesperson problem. This classic combinatorial problem has been studied for years by operations researchers in an attempt to find efficient algorithms to solve it. From 2003 to 2004, a team of researchers tried to find the optimal tour through 24,978 cities, towns, and villages in Sweden, where the distance between any two locations is the Euclidean distance rounded to the nearest integer. A tour was obtained heuristically within a few hours of CPU time and was later proven to be optimal after approximately 84.8 CPU **years** (the exact algorithm was run in parallel on a series of workstations); for information on this and other attempts to solve TSPs, see the book by Applegate et al. [3] and the corresponding TSP web site <http://www.tsp.gatech.edu/>.

Heuristic methods have a rich history. Until the 1950s, when computers were developed and became available, people interested in solving optimization problems mainly developed quick heuristic techniques. Such techniques were all that was available, and many of these methods were effective in finding good solutions.

Starting in the late 1940s, with the development of linear programming, and through the 1960s and 1970s, exact methods for solving problems received more attention. Sophisticated algorithms were developed for a variety of optimization problems, including linear programming and network problems, such as minimum spanning tree, shortest path, and minimum cost network flow problems. What these problems share in common is that each has an *optimality condition* that enables us to determine whether the current solution

is globally optimal. Exact algorithms based on optimality conditions enable us to search for solutions that meet these conditions without having to examine (either explicitly or implicitly) every feasible solution. Problems that have such optimality conditions are often referred to as *nice problems*, for lack of a better term. We will study such exact algorithms for linear programming and network problems in the coming chapters.

Unfortunately, optimality conditions have not been found for all optimization problems. For problems such as general integer or discrete optimization problems, the only known method for finding a global optimal solution is to (essentially) enumerate all possible solutions. As we can imagine, the time and space requirements can be prohibitive. In fact, the amount of time and space spent solving such problems tend to grow exponentially as the number of variables increases. For example, suppose we want to find the maximum value of a function $f(\mathbf{x})$, where each of the n variables satisfy $x_k \in \{0, 1\}$. If we simply enumerate all possible solutions, we would need to examine 2^n possible solutions. If we are able to examine $2^{30} \approx 1$ billion solutions per second, it would still take us about 1 day to solve this problem for $n = 46$ and roughly 1 year to solve a problem for $n = 55$; however, heuristic algorithms can find near-optimal solutions in a fraction of a second, even for larger values of n . For many discrete optimization problem algorithms, we quickly find an optimal solution, but then spend a large amount of time verifying its optimality. In Chapter 14, we study approaches to solving integer programs, including techniques for decreasing the amount of time spent solving for the optimal solution.

5.2 WHAT TO ASK WHEN DESIGNING ALGORITHMS?

Now that we have seen some motivation for the use of heuristics, let's get some experience with designing algorithms for optimization problems, which will help us when we start searching for exact algorithms. When we start to design an algorithm for an optimization problem, we are looking for a sequence of steps that will end with a “reasonable” solution to the problem.

But how do we start? Knowing a little about the problem helps, of course, but there are some fundamental questions we should consider before we actually dive in; this is especially true when trying to derive exact methods. When we attempt to find an algorithm to solve a class of optimization problems, we should ask ourselves the following questions:

1. Is there an optimal solution? Is there even a feasible solution?

These questions are not as simple as they look. For example, suppose we wanted to find the largest value of x such that $x < 4$, that is,

$$\begin{aligned} \max \quad & x \\ \text{s.t. } & x < 4. \end{aligned}$$

This simple optimization problem has no optimal solution. This is one reason why we assume that all inequality constraints are either \leq or \geq . However, we still may not have an optimal solution. For discrete optimization problems, typically there are only a finite (but very large) number of feasible solutions; hence, an optimal solution would obviously exist for these problems.

2. If there is an optimal solution, how do we know if my current solution is one? Can we characterize mathematically what an optimal solution looks like?

We are often interested in developing an *optimality condition* that indicates whether our current solution is optimal. Unfortunately, for many discrete problems, such a condition does not exist. However, for those problems in which we have one, we can design an algorithm in which the optimality condition guides our sequence of examined solutions.

3. Is there more than one optimal solution, and how can we identify the others?

These questions are important because we will assume throughout that an optimization problem will either be unbounded or have an optimal solution. Thus, we need to characterize each case for our algorithm. However, this is not straightforward. For example, suppose we are asked to find the value x that maximizes $f(x) = (x^3 - 2x^2 + 1)e^{-2x^2} \cos 4x$ without the aid of a computer or any graphing calculator. We would probably take the derivative of $f(x)$, set it equal to 0, and solve for x . However, if we do this, we may not end up with the maximizing x . Instead, we may end up

with a *local optimum*, a point that is either higher or lower than points “close” to it. Unfortunately, this does not imply that we have an optimal solution. Thus, we have to consider different types of optimal solutions and what that means for our overall method.

4. If we are not at an optimal solution, how can we get to a solution better than our current one?

This is the fundamental question in designing optimization algorithms, and it is often tied to the idea from question 3—identifying characteristics of an optimal solution.

5. How do we start an algorithm? At what solution should we begin?

It would make sense to start with a feasible solution, but how do we find one?

We consider each of these questions. While some of these topics are more useful in designing exact methods, questions such as 3, 4, and 5 are valid for all algorithms.

5.3 CONSTRUCTIVE VERSUS LOCAL SEARCH ALGORITHMS

As we’ve seen earlier, optimization algorithms can be classified as either exact or heuristic. These classes depend upon whether the algorithm is guaranteed to return an optimal solution (if one exists). We can also partition our algorithms by how the final solution is derived.

Constructive Algorithm An algorithm that constructs a final solution by building upon a partial (incomplete) solution as it iterates is called a *constructive algorithm*. At each iteration, there may be variables that have not been assigned any value.

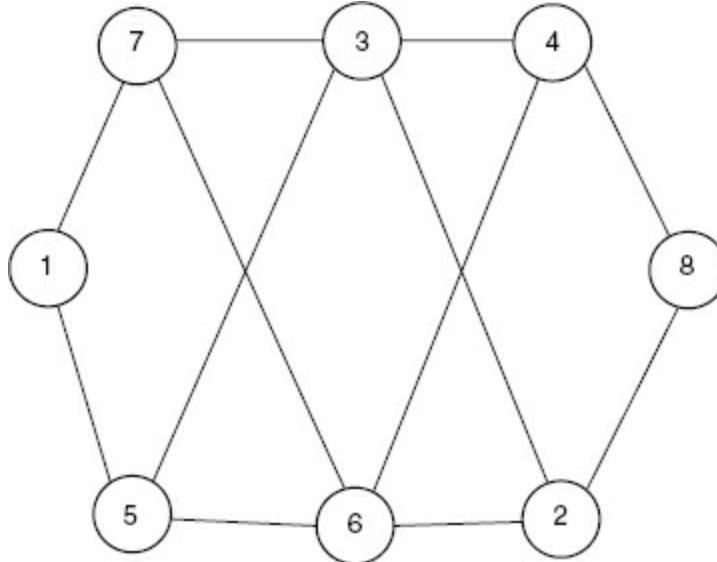
Local Search Algorithm A *local search algorithm* constructs a final solution by starting from some initial (complete) feasible solution and iteratively modifying the current solution to obtain a new one until a better solution is obtained in this manner.

Let's illustrate each of these approaches using the following example.

■ EXAMPLE 5.1

Let's consider the **Graph Coloring Problem**: Given a graph $G = (V, E)$, assign a positive integer c_i (called a *color*) to each node $i \in V$ in such a way that (a) if i and j are connected by an edge $(i, j) \in E$, then $c_i \neq c_j$ and (b) the number of distinct integers used is minimized. An assignment of colors to nodes that satisfies (a) is called a *valid coloring*. To illustrate this problem, consider the graph G given in [Figure 5.1](#).

[FIGURE 5.1](#) Graph coloring example.



A constructive algorithm for this problem could start with no colors assigned to any of the nodes and then iteratively assign to a node the smallest color not assigned to any of its neighboring nodes $N(i) = \{j : (i, j) \in E\}$. Thus, in [Figure 5.1](#), if we start with node 1, we can assign it color $c_1 = 1$; note that we can next assign color $c_2 = 1$ to node 2 as well. However, when we color node 3, since $(2, 3) \in E$, the smallest color we can assign to it is $c_3 = 2$. We next assign color $c_4 = 1$ to node 4. Note, however, that at node 5 we need to color it with $c_5 = 3$ since node 1 has color $c_1 = 1$ and node 3 has color $c_3 = 2$. Node 6 can be assigned color $c_6 = 2$, and, using a similar argument to that given for node 5, we need to color node 7 with $c_7 = 3$. Finally, we can color node 8 with $c_8 = 2$, and thus we have a coloring of the nodes of G that uses

three distinct colors. [Figure 5.2](#) shows this coloring, with the colors listed in the square boxes.

A local search algorithm for the graph coloring problem could take a coloring for all the nodes and modify the coloring so that (a) it maintains a valid coloring, (b) the number of colors used does not increase, and (c) a coloring is not repeated. If we took the coloring generated by the constructive algorithm above, one possible adjustment would be to change the color of node 1 from $c_1 = 1$ to $c_1 = 2$ (see [Figure 5.3](#)). Note that, if we do this, we can iteratively change the colors of nodes 5 and 7 each to $c_5 = c_7 = 1$ since each of their neighbors has color 2 (see [Figure 5.4](#)). It is easy to verify that this is the minimum number of colors (since we have an edge in our graph, at least two colors must be used).

FIGURE 5.2 Coloring using constructive algorithm.

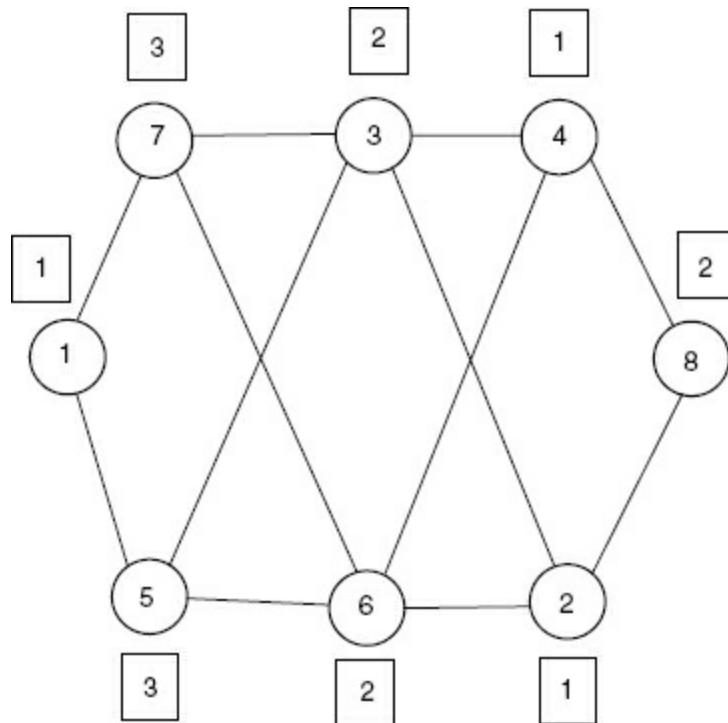
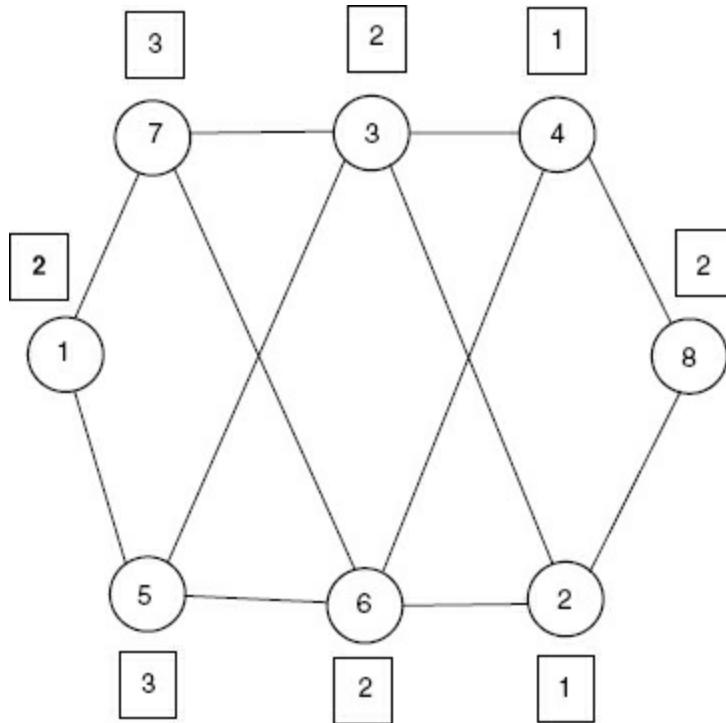
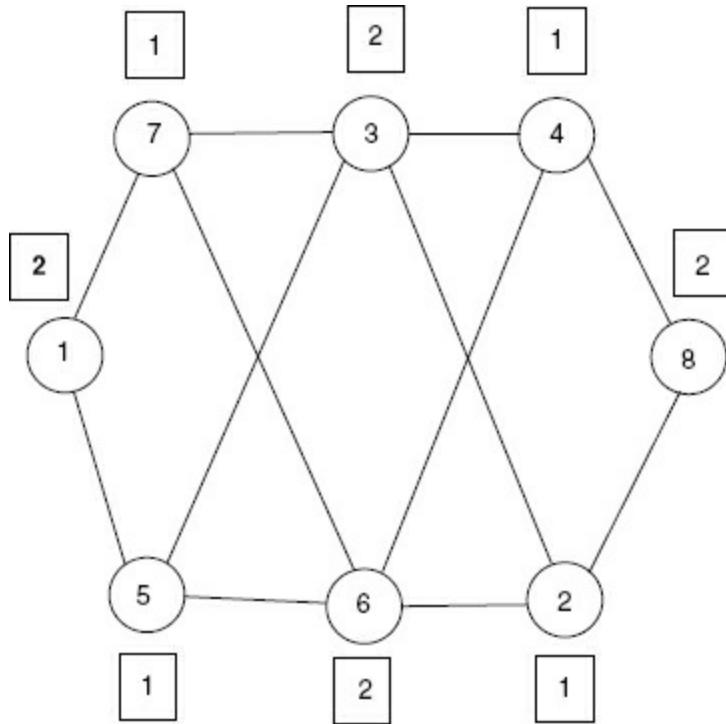


FIGURE 5.3 Coloring using local search algorithm—first iteration.



While heuristics (and exact methods) are problem specific, there are some general principles used when they are designed. For constructive methods, these basic principles can be divided into three parts.

FIGURE 5.4 Coloring using local search algorithm—final.



1. *Incremental Approach*: Each part of the solution is determined incrementally and not collectively. In this way, a solution is built one element at a time, starting from nothing, until a complete solution is generated.
2. *Selection*: Each iteration improves upon the old one, where improvement means increase for a maximization problem and decrease for a minimization problem. Those algorithms that choose from among the available options the one that yields the greatest improvement are referred to as **greedy algorithms**. For example, our constructive algorithm for the graph coloring problem in Example 5.1 is greedy since at each node we select the smallest possible color to assign.
3. *No Backtracking*: Once a part of the solution has been determined, it cannot be “undone” that is, it cannot be removed from the solution and replaced by another. For example, in 5.1, once a node has been assigned a color, it cannot be reassigned a different color.

Because backtracking is not allowed, constructive methods are “single-pass algorithms.” In addition, the selection principle is often what distinguishes one algorithm from another.

Local Search Methods For local search methods, its approach centers around the determination of possible “next solutions.” Typically, we want these solutions to be “close” to the current one. But which solutions are “close?”

Neighborhood Given a solution \mathbf{x} , the *neighborhood* $N(\mathbf{x})$ of \mathbf{x} is the set of all solutions that are close to it, where distance is measured by some predefined metric. By changing our definition of distance, we can change the neighborhood of \mathbf{x} .

■ EXAMPLE 5.2

Let’s consider the feasible solutions represented by the corner points of the three-dimensional cube shown in [Figure 5.5](#). These solutions can be labeled as the 0–1 vectors

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

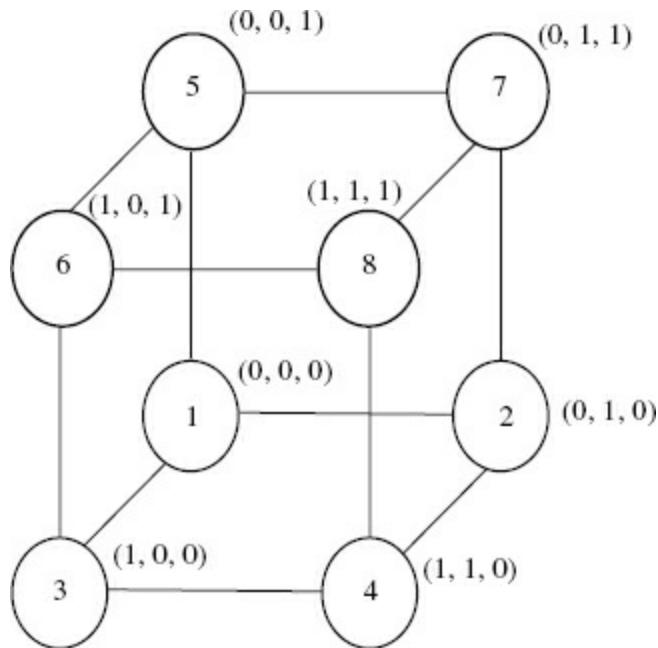
One neighborhood of $\mathbf{x} \in S$ may be all other solutions $\mathbf{y} \in S$ that have all coordinates but one with identical values, that is, where there is an index k

such that $y_i = x_i$ if $i \neq k$ and $y_k = 1 - x_k$. Consider the solution $\mathbf{x} = (0, 0, 1)$, corresponding to vertex 5 on the cube in [Figure 5.5](#). If $N_1(\mathbf{x})$ represents all such neighbors of \mathbf{x} , then

$$N_1((0, 0, 1)) = \{(0, 0, 0), (1, 0, 1), (0, 1, 1)\}.$$

Note that each of these neighbors is connected to \mathbf{x} by an edge of the cube (vertices 1, 6, and 7 on the cube). Another possible neighborhood of \mathbf{x} is the solutions \mathbf{y} where there is an index k such that $y_i = x_i$ if $i \neq k$ and $y_k = 1$ and

[FIGURE 5.5](#) Three-dimensional cube.



$x_k = 0$. If $N_2(\mathbf{x})$ represents all such neighbors of \mathbf{x} , then

$$N_2((0, 0, 1)) = \{(1, 0, 1), (0, 1, 1)\}.$$

These are vertices 6, and 7 on the cube.

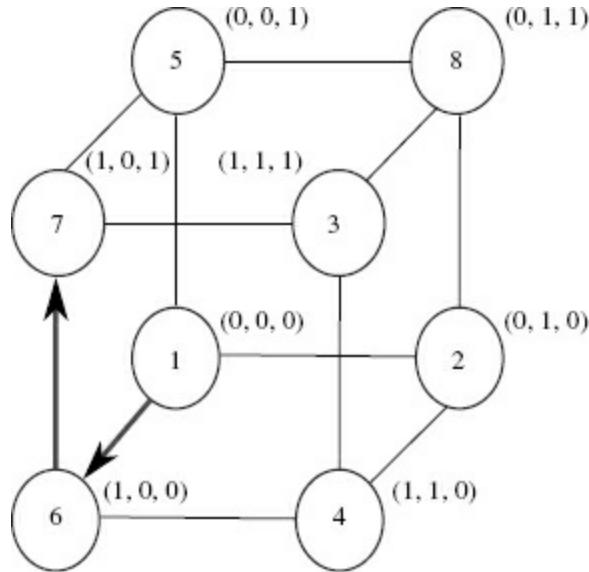
Because of their use of neighborhoods, local search algorithms are sometimes referred to as *neighborhood search algorithms*. Such methods move from solution to solution until the neighborhood of the current solution does not contain a solution that improves upon our current one. It should be noted that local search methods also maintain properties 2 (selection) and 3 (no backtracking) described earlier for constructive methods. The differences among local search algorithms for the same problem occur in both the definition of the neighborhood and the selection criteria used. Of course, there are many ways to select the next solution, ranging from greedy

approaches to purely random choices. Each have advantages and disadvantages, however, and so there is no consensus on which criterion is the best in most situations. In fact, local search methods, depending on their selection criteria, may finish with “locally” optimal solution instead of “globally” optimal ones.

■ EXAMPLE 5.3

Consider the optimization problem on the three-dimensional cube given in [Figure 5.6](#), where the labels of each node indicate its objective value. If we start with the solution $(0, 0, 0)$ and consider only moving toward neighboring solutions joined by an edge of the cube (using neighborhood $N_1(\mathbf{x})$ from Example 5.2), a greedy selection at each step moves us first to $(1, 0, 0)$ and then to $(1, 0, 1)$. Note that there is a path of neighboring solutions that ends at the globally optimal solution $(1, 1, 1)$.

FIGURE 5.6 Greedy local search on three-dimensional cube.



Constructive algorithms are typically faster than local search algorithms, since there are often fewer choices to consider at each iteration, but they often produce weaker solutions. In fact, many approaches first employ a constructive method and then use a local search method to improve it. We should note that it was fortunate that our local search approach found the optimal coloring for the graph in [Figure 5.1](#); this is not always the case. However, there are cases where both constructive algorithms and local search

methods are guaranteed to finish with an optimal solution (if one exists for their problem) and we will see such examples in Section 5.8 and in Chapters 8 and 12.

5.4 HOW GOOD IS OUR HEURISTIC SOLUTION?

We generally want to know how close a heuristic solution value is to that of the global optimal solution. This is a difficult question to answer fully since we do not know the optimal value (of course, if we did know the optimal solution, or could easily generate it, we would not need the heuristic algorithm). However, we can gain information by considering a “relaxed” version of the problem.

Relaxation Given an optimization problem

$$P = \max / \min \{f(\mathbf{x}) : \mathbf{x} \in S\},$$

a *relaxation* P_R of P , given by

$$P_R = \max / \min \{g(\mathbf{x}) : \mathbf{x} \in S'\},$$

is one where every feasible solution of P is also feasible to P_R (so that $S \subseteq S'$) and, for every feasible solution \mathbf{x} in P , the value of \mathbf{x} in P_R is “better” than the value of \mathbf{x} in P . Thus, if P is a maximization problem, then “better” means “at least as large”; if P is a minimization problem, then “better” means “no greater than.”

■ EXAMPLE 5.4

Consider the integer program

$$\begin{aligned}
\max \quad & 8x + 7y \\
\text{s.t.} \quad & -18x + 38y \leq 133 \\
& 13x + 11y \leq 125 \\
& 10x - 8y \leq 55 \\
(5.1) \quad & x, y \geq 0, \text{ integer.}
\end{aligned}$$

If we remove the integrality constraints from each of the variables, we form the linear program

$$\begin{aligned}
\max \quad & 8x + 7y \\
\text{s.t.} \quad & -18x + 38y \leq 133 \\
& 13x + 11y \leq 125 \\
& 10x - 8y \leq 55 \\
(5.2) \quad & x, y \geq 0,
\end{aligned}$$

which is a relaxation of (5.1). If we change the objective of (5.2) to $10x + 7y$, we still have a relaxation, but changing it to $7x + 7y$ does not generate a relaxation since some feasible solutions to (5.1) would have smaller value in (5.2).

Example 5.4 illustrates a common approach to formulating a relaxation; here, we removed some constraints/restrictions placed on the variables. If we remove the integrality constraints from all variables, while leaving the remaining constraints intact, we generate a specific relaxation of the problem.

LP-Relaxation Given an integer program P , the *linear programming relaxation*, or *LP-relaxation*, of P is formed by removing the integrality constraints from all variables while maintaining all upper and lower bounds.

Relaxations can indicate the effectiveness of a heuristic. Suppose that our problem P is a maximization problem, and let P_R be a relaxation of P . If we let $z(P)$ denote the optimal value of P , $z(P_R)$ be the optimal value of P_R , and we suppose that we generated a heuristic (feasible) solution to P with value z_H , then we have the relationship

$$z_H \leq z(P) \leq z(P_R).$$

Thus, we can consider the relaxation gap $z(P_R) - z_H$, which approximates the gap between the heuristic's solution value and the optimal value to P . If this gap is small, then our heuristic solution is close to (or is) the optimal solution. We illustrate this principle for various examples of heuristic algorithms throughout this chapter.

5.5 EXAMPLES OF CONSTRUCTIVE METHODS

Now that we've discussed various algorithmic paradigms, let's move to design. It's important for us as practicing operations researchers to be comfortable with the notion of designing heuristic algorithms for optimization problems, since we often deal with models that require too much computational time to solve exactly. In this section, we describe three common optimization problems and discuss constructive methods for finding good solutions. None are exact methods, but all are based upon reasonable selection criteria.

Heuristic for 0–1 Knapsack Problem

We first look at an example of how to construct a heuristic for a “simple” integer programming problem:

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{s.t.} \quad & a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b \\ (5.3) \quad & x_i \in \{0, 1\}, \quad i \in \{1, \dots, n\}. \end{aligned}$$

This problem is called the **0–1 knapsack problem** due to its interpretation of filling a knapsack of limited volume b with a collection of items, each with volume a_i and “value” c_i , in such a way that we maximize the total value. In our discussions, when we talk about the knapsack, we are referring to the constraint of (5.3), and the right-hand side is the capacity of the knapsack. We shall assume that each $c_i, a_i > 0$, and that $a_i \leq b$ for all i . While this

problem can be computationally difficult for certain values of c_i and a_i , it is typically one of the easier “difficult” problems, and there are algorithms that can often solve many (but not all) instances of this problem efficiently. In fact, knapsack problems are often subproblems in algorithms for more difficult problems.

We begin the design of our constructive algorithm by deciding how to add stuff to our knapsack, so as not to exceed the capacity b . To simplify matters, let’s suppose that S denotes the set of indices that have already been selected (i.e., $x_i = 1$ for all $i \in S$). Note that the amount of room available in the knapsack is

$$\hat{b} = b - \sum_{i \in S} a_i.$$

Two selection techniques are natural:

Selection Criterion 1: Select the next item to be placed into the knapsack as the item i that fits into the knapsack (i.e., $a_i \leq \hat{b}$) and has the largest value c_i .

Selection Criterion 2: Select the next item to be placed into the knapsack as the item i that has the smallest size a_i .

While the second may not appear productive, since the quantity we’re trying to maximize does not play a role in the selection, the first technique seems reasonable. In fact, we can easily create an algorithm based upon this selection criterion. For simplicity, we should keep track of the variables selected (S), the current capacity of the knapsack after items have been selected (\hat{b}), and the items not yet selected that can potentially be placed in the knapsack ($I = \{i : i \notin S \text{ and } a_i \leq \hat{b}\}$).

```

1: Set  $S = \emptyset$ ,  $\hat{b} = b$ ,  $I = \{1, 2, \dots, n\}$ 
2: repeat
3:   Let  $j \in I$  be the index such that  $c_j \geq c_i, i \in I$ .
4:    $I = I - \{j\}$ ,  $S = S \cup \{j\}$ 
5:    $\hat{b} = \hat{b} - a_j$ 
6:   Remove from  $I$  all indices  $i$  where  $a_i > \hat{b}$ 
7: until  $I = \emptyset$ 
```

While this approach makes sense, it can arbitrarily produce bad solutions.

■ EXAMPLE 5.5

Consider the following knapsack problem

$$\begin{aligned}
\max \quad & 10x_1 + 5x_2 + 5x_3 + 5x_4 + 5x_5 + 5x_6 \\
\text{s.t.} \quad & 10x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 \leq 11 \\
& x_1, x_2, x_3, x_4, x_5, x_6 \in \{0, 1\}.
\end{aligned}$$

For the above algorithm, we have $S = \{1\}$, which yields the solution $(1, 0, 0, 0, 0, 0)$ with value of 10. However, it is easy to see that the optimal solution is $(0, 1, 1, 1, 1, 1)$ with a value of 25. Our heuristic solution was not close to the optimal solution.

You may have noticed in this example that the variables x_2, x_3, x_4, x_5, x_6 all gave “more bang for the buck” than x_1 , in that the value to cost ratio $\frac{5}{2}$ of was much better than $\frac{10}{10}$. This leads to a third selection criterion.

Selection Criterion 3: Select as the next item to be placed into the knapsack the item i that both fits into the knapsack ($a_i \leq b$) and has the largest ratio $\frac{c_i}{a_i}$. In the case of tie, choose any of the variables that have this largest value.

In fact, this selection criterion has a basis from the LP-relaxation of the knapsack problem. It can (and will later) be shown that if the variables are ordered so that

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \dots \geq \frac{c_n}{a_n},$$

then the optimal solution to the LP-relaxation (where $x_j \in \{0, 1\}$ is replaced by $0 \leq x_j \leq 1$) is given by

$$x_k = \begin{cases} 1, & k < r, \\ \frac{b - \sum_{i=1}^{r-1} a_i}{a_r}, & k = r, \\ 0, & k > r, \end{cases}$$

where r is the unique index satisfying

$$\sum_{i=1}^{r-1} a_i \leq b \quad \text{and} \quad \sum_{i=1}^r a_i > b.$$

For example, consider the LP

$$\begin{aligned} \max \quad & 8x_1 + 5x_2 + 9x_3 + 10x_4 + 5x_5 \\ \text{s.t.} \quad & \end{aligned}$$

$$2x_1 + 2x_2 + 4x_3 + 5x_4 + 3x_5 \leq 12$$

$$x_1, x_2, x_3, x_4, x_5 \leq 1$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0,$$

which has as an optimal solution of $(1, 1, 1, \frac{4}{5}, 0)$. We modify our original algorithm to that given in Algorithm 5.1.

Algorithm 5.1 Greedy Heuristic for 0-1 Knapsack Problems

Order the variables so that $\frac{c_i}{a_i} \geq \frac{c_{i+1}}{a_{i+1}}$ for $i = 1, 2, \dots, n - 1$.

Set $\hat{b} = b$, $S = \emptyset$.

for all i from 1 to n **do**

if $a_i \leq \hat{b}$ **then**

$S = S \cup \{i\}$.

 Set $\hat{b} = \hat{b} - a_i$.

end if

end for

The solution x is then given by

$$x_i = \begin{cases} 1, & \text{if } i \in S, \\ 0, & \text{otherwise.} \end{cases}$$

■ EXAMPLE 5.6

Consider the 0–1 knapsack problem

$$\max \quad 41x_1 + 43x_2 + 15x_3 + 35x_4 + 33x_5 + 31x_6 + 29x_7$$

s.t.

$$19x_1 + 20x_2 + 7x_3 + 18x_4 + 17x_5 + 16x_6 + 15x_7 \leq 61$$

$$x_i \in \{0, 1\}.$$

Note that the variables are already ordered in terms of the ratios $\frac{c_i}{a_i}$. Algorithm 5.1 gives

$\hat{b} = 61$.
 $i = 1$: Since $a_1 = 19 < \hat{b}$, $S = \{1\}$, $\hat{b} = 61 - 19 = 42$.
 $i = 2$: Since $a_2 = 20 < \hat{b}$, $S = \{1, 2\}$, $\hat{b} = 42 - 20 = 22$.
 $i = 3$: Since $a_3 = 7 < \hat{b}$, $S = \{1, 2, 3\}$, $\hat{b} = 22 - 7 = 15$.
 $i = 4$: Since $a_4 = 18 > \hat{b}$, $S = \{1, 2, 3\}$.
 $i = 5$: Since $a_5 = 17 > \hat{b}$, $S = \{1, 2, 3\}$.
 $i = 6$: Since $a_6 = 16 > \hat{b}$, $S = \{1, 2, 3\}$.
 $i = 7$: Since $a_7 = 15 = \hat{b}$, $S = \{1, 2, 3, 7\}$, $\hat{b} = 15 - 15 = 0$.

We end with the feasible solution $(1, 1, 1, 0, 0, 0, 1)$ with a value of 128. Note that if we had used selection criterion 1, we would have ended with the feasible solution $(1, 1, 0, 1, 0, 0, 0)$ with value 119.

If we remove the integrality restriction from each of the variables and replace them with $0 \leq x_i \leq 1$, then we find that the LP-relaxation has an optimal value of $\frac{128}{6}$. Since all our objective coefficients are integers, we know that the optimal integer solution must have an integer value, and hence, our heuristic has found the optimal solution. However, if the objective coefficient of variable x_7 is changed from 29 to 25, our LP-relaxation optimal value remains the same, but our heuristic solution's value is now 124. The relaxation gap in this case would have been $\frac{128}{6} - 124 = \frac{4}{6}$, indicating that it may be possible to find a better solution; however, even an improved solution would be only slightly better than the one found.

Heuristic for Set Covering Problems

Recall from Section 3.2 that the set covering problem is

$$\begin{aligned}
 \min \quad & \sum_{i=1}^n c_i x_i \\
 \text{s.t.} \quad & \sum_{i \in S_j} x_i \geq 1, \quad j \in \{1, 2, \dots, m\} \\
 & x_i \in \{0, 1\}, \quad i \in \{1, 2, \dots, n\},
 \end{aligned}$$

where each $S_j \subset \{1, 2, \dots, n\}$. We assume that there are not two subsets S_j and S_k in $\{1, 2, \dots, n\}$ such that $S_j \subseteq S_k$. If this were the case, then we can

eliminate the constraint corresponding to S_k (see Exercise 5.6). In addition, we assume that each $c_i > 0$. The following defines the fundamental entities we are looking for.

A variable x_i is said to **cover** the j th constraint if $i \in S_j$. Hence, if $x_i = 1$ in some feasible solution, then the j th constraint is automatically satisfied. A **cover** is a collection of x_i that collectively satisfy the constraints.

Let's first consider the case where each $c_i = 1$. This version of the problem is referred to as the **Minimum Cardinality Set Covering Problem**. Our aim is to find the smallest cover. We construct a solution to this problem, starting with all variables unassigned, by successively adding a variable $x_i = 1$ to our solution until all constraints are covered. At each iteration, our neighborhood around the current “solution” \mathbf{x} would be identical to that for the 0–1 knapsack problem in Section 5.5, in that each solution in the neighborhood has (a) a 1 in position i if $x_i = 1$, and (b) the number of positions with 1 is one more than the total number of 1's in \mathbf{x} .

Since we want a smallest cover, it seems reasonable that each variable selected should cover as many currently uncovered constraints as possible. This leads to the following search.

Selection Criterion for Set Covering Heuristic: For each unselected variable x_j ($x_j = 0$ in the current partial solution), let d_j denote the number of uncovered constraints that would be covered by variable x_j . Choose variable i that has the largest value of d_i . In the case of tie, choose any of the variables that have this largest value.

Given this selection criterion, we can write our algorithm as given in Algorithm 5.2.

■ EXAMPLE 5.7

Consider the set covering problem first given in Section 3.2.

$$\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \geq 1 \\
& x_3 + x_4 \geq 1 \\
& x_1 + x_4 \geq 1 \\
& x_2 + x_4 + x_5 \geq 1 \\
& x_1 + x_3 \geq 1 \\
& x_1 + x_5 \geq 1 \\
& x_2 + x_4 \geq 1 \\
& x_3 + x_6 \geq 1 \\
& x_2 + x_6 \geq 1 \\
& x_5 + x_6 \geq 1 \\
& x_i \in \{0, 1\}, \quad i \in \{1, 2, \dots, 6\}.
\end{aligned}$$

Algorithm 5.2 Heuristic for Minimum Cardinality Set Covering Problem

Set $T = \emptyset$ (variables selected).

while There are uncovered constraints **do**

 For each $j \notin T$, let d_j = number of uncovered constraints covered by x_j .

 Let i be the index yielding the largest value of d_j , $j \notin T$; that is,

$$i = \arg \max_{j \notin T} d_j.$$

 If a tie exists, choose the index i randomly from those with largest value of d_j .

$T = T \cup \{i\}$.

 Remove those constraints covered by x_i .

end while

Solution \mathbf{x} is then given by

$$x_i = \begin{cases} 1, & \text{if } i \in T, \\ 0, & \text{otherwise.} \end{cases}$$

In the first iteration, we have

$$\mathbf{x} = (0, 0, 0, 0, 0, 0)$$

$$\mathbf{d} = (4, 4, 3, 4, 3, 3).$$

Since there is a tie among x_1, x_2 , and x_4 , we randomly choose x_2 . This leaves the following uncovered constraints:

$$\begin{array}{lll}
& x_3 + x_4 & \geq 1 \\
x_1 & + x_4 & \geq 1 \\
x_1 & + x_3 & \geq 1 \\
x_1 & + x_5 & \geq 1 \\
x_3 & + x_6 & \geq 1 \\
& x_5 + x_6 & \geq 1,
\end{array}$$

with $d_1 = 3, d_3 = 3, d_4 = 2, d_5 = 2$, and $d_6 = 2$. At the second iteration, given the tie between x_1 and x_3 , we randomly choose x_3 . This leaves the uncovered constraints:

$$\begin{array}{lll}
x_1 & + x_4 & \geq 1 \\
x_1 & + x_5 & \geq 1 \\
x_5 + x_6 & \geq 1,
\end{array}$$

with $d_1 = 2, d_4 = 1, d_5 = 2$, and $d_6 = 1$. At the third iteration, we choose randomly x_1 (from x_1 and x_5), which leaves the uncovered constraint

$$x_5 + x_6 \geq 1.$$

As the fourth iteration, we choose either x_5 or x_6 (let's choose x_5), which gives the feasible solution $(1, 1, 1, 0, 1, 0)$. Note that this solution is not optimal since $(1, 0, 0, 1, 0, 1)$ is also feasible.

We can modify this algorithm to find a good feasible solution to the general problem where each $c_i > 0$ (see Exercise 5.5).

Heuristics for Traveling Salesperson Problems

In Section 3.4, we saw that the traveling salesperson problem (TSP for short) was one of the classic operations research and discrete optimization problems. The problem consists of n cities and a distance c_{ij} of traveling from city i to city j . The goal is to determine a minimum distance ordering (or tour) of the cities where each city i is visited exactly once, beginning and ending with the same city. This sequencing problem can be stated as an integer program but is easier to work with heuristically in a combinatorial manner.

Many constructive heuristics attempt to construct a near-optimal tour by

building it one city at a time. Two ways this can be done are (1) constructing a sequence that is not a tour until all cities have been added and (2) constructing a sequence in such a way that the cities already added define a “subtour” and that a complete tour is generated only when the last city is added. By a subtour, we mean a tour that goes through only a subset of the given cities. For example, if we had $n = 7$ cities, one subtour would be $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$. We look at designing heuristics using both techniques.

Nearest Neighbor Heuristic Let's consider a heuristic that only constructs a complete tour at the end and no subtours during the construction. In this case, we add a city to the last city visited. For example, if we've already identified the sequence of cities as $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, we'll next identify which city to visit after city 4. The following selection criterion is appropriate.

Selection Criteriaon 1: The next city in the sequence is the unselected city that is closest to the current city.

■ EXAMPLE 5.8

Suppose that we have identified the sequence of cities $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ in a seven-city TSP; see [Figure 5.7](#). From city 4, we have the following distances:

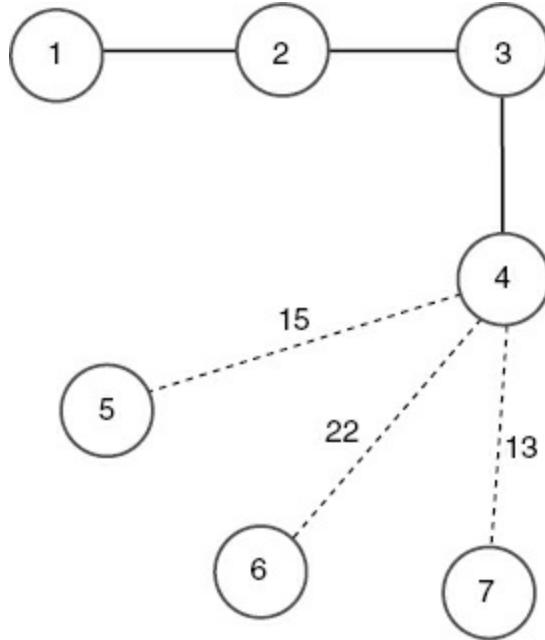
$$d_{45} = 15$$

$$d_{46} = 22$$

$$d_{47} = 13.$$

Since city 7 is closest to city 4, we would add city 7 to the sequence, yielding $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 7$.

[FIGURE 5.7](#) Example of nearest neighbor selection.



To get a variety of tours, this process is typically repeated with each city selected as the initial city. The complete nearest neighbor heuristic based upon selection criteriaon 1 is given in Algorithm 5.3.

■ EXAMPLE 5.9

Consider the following six-city tour with distance matrix

$$c = \begin{bmatrix} - & 16 & 23 & 14 & 8 & 15 \\ 16 & - & 12 & 19 & 9 & 13 \\ 23 & 12 & - & 7 & 25 & 16 \\ 14 & 19 & 7 & - & 18 & 15 \\ 8 & 9 & 25 & 18 & - & 20 \\ 15 & 13 & 16 & 15 & 20 & - \end{bmatrix}.$$

If we run Algorithm 5.3, for $i = 1$, we see that the closest neighbor to 1 that has not been selected is 5, with a distance of 8. Thus, we add 5 to S and find the closest unselected neighbor to 5, which is 2. Adding 2 to S , we now look at the closest neighbor to 2 among the cities in $\{3, 4, 6\}$; since 3 is the closest with distance 12, we add 3 to S . Searching for the closest neighbor of 3 from among $\{4, 6\}$, we see that 4 is the closest. Since 6 is the last remaining unselected city, this gives us a tour of $1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 1$ with total distance $8 + 9 + 12 + 7 + 15 + 15 = 66$. Repeating this for each city, we obtain the following tours:

Algorithm 5.3 Nearest Neighbor Heuristic for TSP

Given: n cities and cost matrix $c = (c_{ij})$.
for all Cities $i = 1 \dots n$ do
 $V = \{1, 2, \dots, n\} - \{i\}$ is the collection of unselected cities.
 $S \leftarrow i$ (initialize sequence to start with city i).
 while $V \neq \emptyset$ do
 Let k be the most recently selected city.
 Let city $j \in \arg \min_{p \in V} c_{kp}$ be a city closest to k . Ties are broken arbitrarily.
 Add j to the end of S .
 Let $V = V - \{j\}$.
 end while
 If sequence S is minimum distance found so far, save it.
end for
Return the best sequence found.

Starting City	Tour	Total Distance
1	1 → 5 → 2 → 3 → 4 → 6 → 1	66
2	2 → 5 → 1 → 4 → 3 → 6 → 2	76
3	3 → 4 → 1 → 5 → 2 → 6 → 3	76
4	4 → 3 → 2 → 5 → 1 → 6 → 4	66
5	5 → 1 → 4 → 3 → 2 → 6 → 5	74
6	6 → 2 → 5 → 1 → 4 → 3 → 6	76

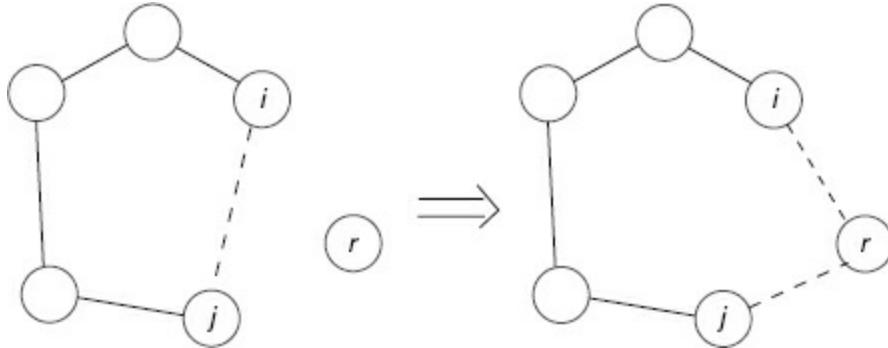
The best tour is $1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 1$, with total distance of 66.

Cheapest Insertion Heuristic Another heuristic for the TSP constructs a tour by maintaining a subtour throughout each iteration. To see how we can construct such an algorithm, suppose we have a subtour and are looking to add city r to the subtour. One way to add r is to identify a link (i, j) to eliminate, so that we can add the links (i, r) and (r, j) ; see [Figure 5.8](#).

One possible selection criterion is the following.

Selection Criterion: Given a subtour S , find a link (i, j) and a city r not in S that minimizes $c_{ir} + c_{rj} - c_{ij}$, that is, minimizes the distances added to the subtour.

FIGURE 5.8 Example of insertion method for finding tour.



We need to initialize this heuristic by first selecting two cities to be our initial subtour; this can be done by first selecting an initial city and then finding its closest neighbor. Like the nearest neighbor heuristic, we typically run this algorithm with each city as the initial city and choose the best tour found. A formal version of the cheapest insertion heuristic is given in Algorithm 5.4.

Algorithm 5.4 Cheapest Insertion Heuristic for TSP

Given: n cities and cost matrix $\mathbf{c} = (c_{ij})$.
for all Cities $k = 1 \dots n$ **do**
 Let $V = \{1, 2, \dots, n\} - \{k\}$ be the unselected cities.
 Let $l \in \arg \min_{j \in V} c_{kj}$, with ties broken arbitrarily.
 $S \leftarrow (k \rightarrow l \rightarrow k)$ (initialize sequence to start with subtour $k \rightarrow l \rightarrow k$).
 Let $V = V - \{l\}$.
 while $V \neq \emptyset$ **do**
 Find $r \in V$ and $i \rightarrow j$ link in S that minimizes $c_{ir} + c_{rj} - c_{ij}$.
 Add r between i and j in S , removing link $i \rightarrow j$.
 Let $V = V - \{r\}$.
 end while
 If sequence S is minimum distance found so far, save it.
end for
 Return the best sequence found.

■ EXAMPLE 5.10

Consider the following five-city tour with distance matrix

$$\mathbf{c} = \begin{bmatrix} - & 16 & 23 & 16 & 8 \\ 16 & - & 12 & 19 & 25 \\ 23 & 12 & - & 15 & 10 \\ 16 & 19 & 15 & - & 18 \\ 8 & 25 & 10 & 18 & - \end{bmatrix}.$$

If we start the algorithm at city 1, we first look for the closest city to it, which is city 5, and get the initial subtour $1 \rightarrow 5 \rightarrow 1$ with cost $8 + 8 = 16$. To insert the next city, we compare the following subtours by their additional costs:

Subtour	Additional Cost
$1 \rightarrow 2 \rightarrow 5 \rightarrow 1$	$16 + 25 - 8 = 33$
$1 \rightarrow 5 \rightarrow 2 \rightarrow 1$	$25 + 16 - 8 = 33$
$1 \rightarrow 3 \rightarrow 5 \rightarrow 1$	$23 + 10 - 8 = 25$
$1 \rightarrow 5 \rightarrow 3 \rightarrow 1$	$10 + 23 - 8 = 25$
$1 \rightarrow 4 \rightarrow 5 \rightarrow 1$	$16 + 18 - 8 = 26$
$1 \rightarrow 5 \rightarrow 4 \rightarrow 1$	$18 + 16 - 8 = 26$

Since there are different subtours with lowest additional cost, we choose randomly the subtour $1 \rightarrow 3 \rightarrow 5 \rightarrow 1$ that has cost $16 + 25 = 41$. From here, we consider another set of subtours and their costs:

Subtour	Additional Cost
$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 1$	$16 + 12 - 23 = 5$
$1 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 1$	$16 + 15 - 23 = 8$
$1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 1$	$12 + 25 - 10 = 27$
$1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$	$15 + 18 - 10 = 23$
$1 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 1$	$25 + 16 - 8 = 33$
$1 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 1$	$18 + 16 - 8 = 26$

We choose the subtour $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 1$, which has cost $41 + 5 = 46$. Finally, to add in city 4, we consider:

Subtour	Additional Cost
$1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 1$	$16 + 19 - 16 = 19$
$1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 1$	$19 + 15 - 12 = 22$
$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$	$15 + 18 - 10 = 23$
$1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 1$	$18 + 16 - 8 = 26$

This gives the complete tour of $1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 1$, with total cost $46 + 19 = 65$.

Relaxation of TSP We'd like to obtain a simple relaxation of the traveling salesperson problem in order to determine how well our heuristic algorithms perform. One such relaxation is the LP-relaxation of the MTZ formulation given in Section 3.4:

$$\begin{aligned}
\min \quad & \sum_i \sum_j c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_j x_{ij} = 1, \quad i \in V \\
& \sum_j x_{ji} = 1, \quad i \in V \\
& u_i - u_j + 1 \leq (n-1)(1-x_{ij}), \quad i, j \in V - \{1\}; i \neq j \\
& u_1 = 1 \\
& 2 \leq u_i \leq n, \quad i \in V - \{1\} \\
& 0 \leq x_{ij} \leq 1, \quad i, j \in V.
\end{aligned}$$

For the problem in Example 5.9, this relaxation gives an optimal value of 58.8, which is less than our heuristic solution of 66. Note that this is not a good relaxation to our problem since the optimal value to this particular TSP is 66 (our heuristic solution is optimal in this case). For the problem in 5.10, the relaxation has an optimal value of 62. This is a decent relaxation in this case since the optimal tour has a value of 65 (again, our heuristic generated the optimal tour).

In order to solve large TSP instances, better relaxations and heuristic algorithms are needed. A recent book by Applegate et al. [3] discusses their approach to solving large TSP problems. One such example consists of over 1.9 million locations throughout the world. It noted that the best heuristic solution was within 0.05% of the best relaxation value found, but this relaxation took over 1 year of CPU time to solve!

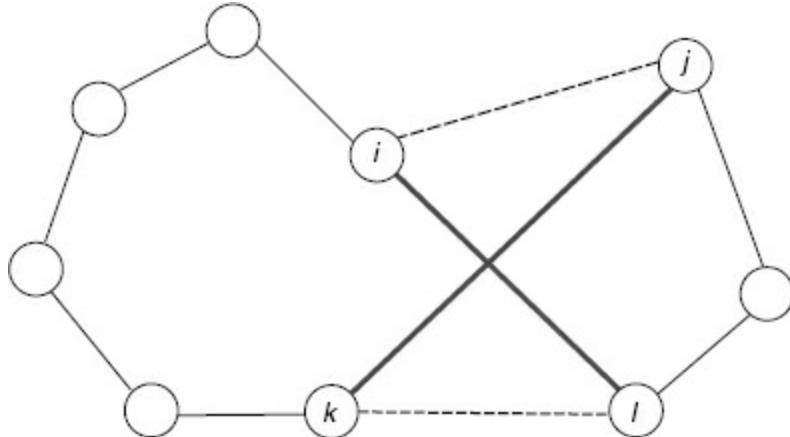
5.6 EXAMPLE OF A LOCAL SEARCH METHOD

Once a feasible solution has been constructed, it is often worthwhile to try to improve it with a local search algorithm. A common technique is to exchange some components in the solution with some not in the solution. Such **interchange methods** continue until no improvement is possible. The

solution found is a **local optima**, where all solutions in the corresponding neighborhood are not any better. Of course, local optima are not necessarily global optima, but they are better than other solutions “close by.”

How these interchange heuristics work is easily described with an example of the TSP. Suppose we have a tour and consider two nonadjacent edges (i, j) and (k, l) of this tour. If these edges are removed, the tour separates into two paths T_1 and T_2 that can be recombined into a new tour T' by adding the edges (i, l) and (j, k) (see [Figure 5.9](#)). If the cost of T is better than that of T' , replace T with T' and continue. Such a process is called a *2-interchange*. If we come to a tour that cannot be improved by a 2-interchange, such a tour is called *2-optimal*; at this point, we stop the interchange heuristic. This heuristic is referred to as *2-opt*. Note that we can do the same for any set of k edges in the tour, producing a *k-opt* heuristic.

FIGURE 5.9 Example of a 2-interchange and its unique new solution.



■ EXAMPLE 5.11

Let’s consider the TSP with cost matrix

$$\mathbf{c} = \begin{bmatrix} - & 16 & 23 & 16 & 8 \\ 16 & - & 17 & 19 & 25 \\ 23 & 17 & - & 15 & 10 \\ 16 & 19 & 15 & - & 18 \\ 8 & 25 & 10 & 18 & - \end{bmatrix},$$

and let an initial tour be $1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 1$. In the first iteration of 2-opt, we would remove and add the edges given below and would have the

following changes in cost:

Remove	Add	Change in Cost
(1, 4), (2, 3)	(3, 4), (1, 2)	$-(16 + 17) + (15 + 16) = -2$
(2, 4), (3, 5)	(2, 5), (3, 4)	$-(19 + 10) + (25 + 15) = 11$
(2, 3), (1, 5)	(2, 5), (1, 3)	$-(17 + 8) + (25 + 23) = 23$
(3, 5), (1, 4)	(1, 3), (4, 5)	$-(10 + 16) + (23 + 18) = 15$
(1, 5), (2, 4)	(1, 2), (4, 5)	$-(8 + 19) + (16 + 18) = 7$

Even though there are 10 pairs of edges to remove, only 5 of them are possible because the remaining 5 pairs are consecutive edges in the tour. Since replacing edges (1, 4), (2, 3) with (3, 4), (1, 2) decreases the cost by 2, we make this change, which results in the new tour $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 1$. We leave it as an exercise to verify that this tour is 2-optimal (see Exercise 5.10).

Unfortunately, such interchange algorithms can take a long time to complete. As an example, for the traveling salesperson problem it was shown by Papadimitriou and Steiglitz [71] that, under some choices of edges to add and remove, an exponential number of interchanges can be made before a 2-optimal solution is obtained. Hence, it is often useful to bound the number of interchanges in practice.

5.7 OTHER HEURISTIC METHODS

While local search methods are common, other heuristic approaches have emerged, and in Chapter 15 we will examine some of these approaches.

One limitation of local search methods is that once they cannot improve upon the current solution, the method stops. Many researchers have tried to overcome this limitation. Some run a heuristic many times, where each time they start at a randomly selected starting point. Others incorporate random elements into local search methods by choosing outside the selection rule. If repeated, the best solution found is used as the final solution. One popular class of algorithms using this methodology are **GRASP algorithms** or greedy randomized adaptive search procedures.

Another class of adaptations of local search methods purposely degrades the current solution in order to diversify the search. One of the best

approaches in this class allows one to make the solution value worse using a probabilistic scheme. This is the basis of **Simulated Annealing Algorithms**.

A similar adaptation is found in **Tabu Search**. These methods try to model human memory by making certain moves “tabu” when leaving a local optimal solution. These moves remain forbidden for a number of steps before becoming active again.

The last class of algorithms we mention are **Genetic Algorithms**. These methods are modeled after biological systems that adapt and evolve into more successful organisms. Here, a population of possible solutions is given initially and then altered by various techniques (mating, mutation, etc.). Some of the new solutions are discarded (die off), while the others carry on populating later generations of solutions.

5.8 DESIGNING EXACT METHODS: OPTIMALITY CONDITIONS

We’ve seen that a common-sense approach to algorithm design, with or without mathematical justification (in the form of some proof of its validity), can achieve great results. However, if we want to design an algorithm that is guaranteed to solve the problem to optimality, some mathematical analysis is required. Something that works for many problems is to identify both necessary and sufficient conditions for a solution to be optimal and use them to derive an algorithm. To illustrate, let’s consider the minimum spanning tree problem from graph theory.

Minimum Spanning Tree Problem Given a connected undirected graph $G = (V, E)$, where we associate with each $(i, j) \in E$ a *length* or *cost* c_{ij} , the *minimum spanning tree problem* is to then find a spanning tree T of G whose total cost

$$\sum_{(i,j) \in T} c_{ij}$$

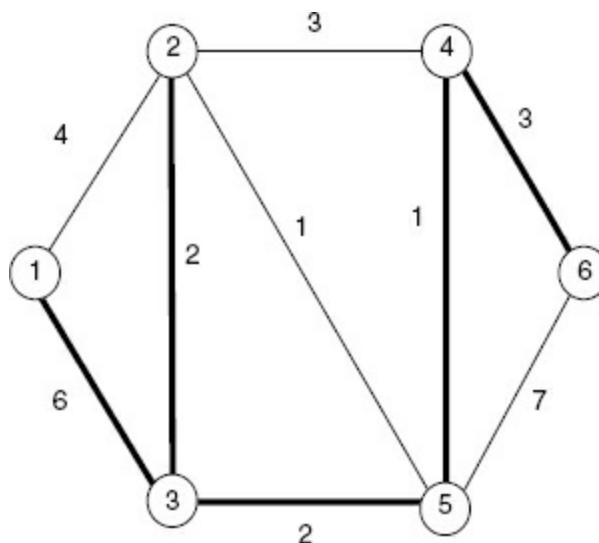
is minimized.

Those edges that are in T are referred to as **tree edges**, while those not in T are referred to as **nontree edges**.

We first saw this problem in Section 3.4 where its integer programming formulation required an exponential number of constraints. Our goal now is to design an exact algorithm for this problem without writing down the integer program.

Throughout this section we assume that all edge costs c_{ij} are integers. This is not necessary, but is reasonable given that we can only represent numbers as rational numbers (even π is represented in any computer program as a rational number), and rational numbers can be multiplied by a large enough integer to make them all integers.

FIGURE 5.10 Tree T from Example 5.12.



In order to determine necessary and sufficient conditions for a problem, it is often useful to examine a specific example and use this to help derive the conditions. To do this, we need to recognize when a given solution is not optimal.

■ EXAMPLE 5.12

Consider the graph given in [Figure 5.10](#), which includes the spanning tree

$$T = \{(1, 3), (2, 3), (3, 5), (4, 5), (4, 6)\}.$$

Is T a minimum spanning tree? It should be obvious that the answer is no

because if we replace the edge $(1, 3)$ with the edge $(1, 2)$, and the edge $(3, 5)$ with the edge $(2, 5)$, generating the new tree

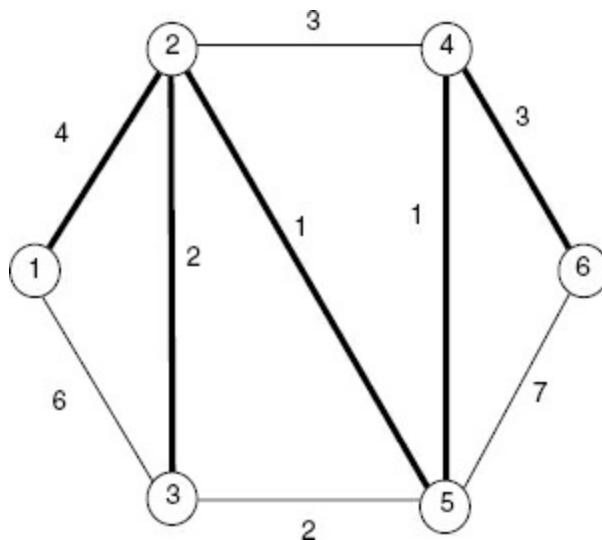
$$T^1 = \{(1, 2), (2, 3), (2, 5), (4, 5), (4, 6)\}$$

we can reduce the total cost of the edges by 3. Is tree T^1 , given in [Figure 5.11](#), now optimal? It seems likely, but how would we know?

Can we determine, based upon this example, a necessary condition for an optimal solution to the minimum spanning tree problem? We saw that the tree $T = \{(1, 3), (2, 3), (3, 5), (4, 5), (4, 6)\}$ cannot be optimal because we can replace the edge $(1, 3)$ with the edge $(1, 2)$ and reduce the cost of the tree. What sort of mathematical condition are we describing here? It helps to know two very important facts about trees:

1. Deleting a tree edge (i, j) partitions the nodes into two subsets S_i and S_j (where $V = S_i \cup S_j$ and $S_i \cap S_j = \emptyset$), such that the subgraphs $G(S_i)$ and $G(S_j)$ containing only tree edges are both connected. The edges (k, l) of G where $k \in S_i$ and $l \in S_j$ is called a *cut* and is denoted by (S_i, S_j) .

[FIGURE 5.11](#) Tree T^1 from Example 5.12.



2. For every nontree edge (k, l) , there exists a unique path on T from k to l . Hence, adding the edge (k, l) to the tree creates a unique cycle.

Note that these two properties can be considered duals of each other, in the sense that they are different yet very similar at the same time. Property 1 deletes a tree edge, and we would then need to add a nontree edge to make a new tree. Property 2 first adds a nontree edge, forcing us to remove a tree

edge to maintain the tree structure. First consider Property 1. In Example 5.12, if we remove the edge $(1, 3)$ from the tree, we have the cut $(\{1\}, \{2, 3, 4, 5, 6\})$. Considering all edges in the cut, we see that the edge $(1, 2)$ has smaller cost than that of edge $(1, 3)$. This suggests that if a tree is a minimum spanning tree, then if we were to delete one of its edges, forming a cut, all other edges in this cut cannot have smaller cost than the one deleted. Formally, this can be stated as follows.

Lemma 5.1 *Given a network G , if a spanning tree T^* is a minimum spanning tree, then, for every tree edge $(i, j) \in T^*$,*

$$c_{ij} \leq c_{kl}$$

for all nontree edges (k, l) contained in the cut formed by deleting the edge (i, j) from T^ .*

Our next question is, “Is this condition sufficient?”, that is, if a spanning tree satisfies the above property, is this spanning tree a minimum spanning tree? If it does, we have generated an **optimality condition** for the minimum spanning tree problem, which determines whether a feasible solution is optimal without comparing it with all other feasible solutions. The answer to this question is yes, giving us what is commonly referred to as the *Cut Optimality Condition*.

Cut Optimality Condition Given a connected undirected network $G = (V, E)$ with edge costs c_{ij} , a spanning tree T^* is a minimum spanning tree if and only if, for every tree edge $(i, j) \in T^*$,

$$(5.4) \quad c_{ij} \leq c_{kl}$$

for all nontree edges (k, l) contained in the cut formed by deleting the edge (i, j) from T^*

.

Proof We shall only prove the sufficient condition since necessity was already given. Assume that T^* is a spanning tree such that, for every tree edge $(i, j) \in T^*$,

$$c_{ij} \leq c_{kl}$$

for all nontree edges (k, l) contained in the cut formed by deleting the edge (i, j) from T^* . Let T^0 be a minimum spanning tree where $T^* \neq T^0$. This implies

that there is an edge $(i, j) \in T^*$ that is not in T^0 . If we delete edge (i, j) from T^* , we create the cut (S_i, S_j) . Note that there is also at least one edge $(k, l) \in T^0$ where $k \in S_i$ and $l \in S_j$ (why?). Replace edge (i, j) in T^* by (k, l) . The resulting tree has a cost that is no less than that of T^* . We can continue this process until the resulting tree is equal to T^0 . Since T^0 is a minimum spanning tree, T^* must have the same cost as T^0 and thus must also be a minimum spanning tree.

Now that we have found a necessary and sufficient optimality condition for the minimum spanning tree problem, how do we derive an exact algorithm to solve these problems? Typically, the most basic of such algorithms explicitly exploit this optimality condition. For example, suppose we have a spanning tree that violates the cut optimality condition (5.4). This implies that there is a tree edge (i, j) and a nontree edge (k, l) in the cut (S_i, S_j) obtained by deleting (i, j) such that $c_{kl} < c_{ij}$. Swapping these edges produces a new tree. We continue this process until we obtain a tree that satisfies the cut optimality condition; hence, it would be a minimum spanning tree. Formally, this algorithm is given in Algorithm 5.5. Note that this is a local search algorithm since we start with a feasible solution and iteratively improve it until the algorithm stops; unlike previous such algorithms, Algorithm 5.5 is guaranteed to end in an optimal solution.

Algorithm 5.5 Generic Cut Optimality Algorithm

Find a spanning tree T of G .
while There exist edges $(i, j) \in T$ and (k, l) in cut (S_i, S_j) violating(5.4) do
$T \leftarrow T \cup \{ (k, l) \} - \{ (i, j) \}$.
end while

Can we also find an optimality condition based upon Property 2 of trees? If we were to add a nontree edge (k, l) to our tree, we'd form a unique cycle. We'd need to remove one of the edges from this cycle in order to get a tree back. If we're trying to find a minimum spanning tree, we'd remove the edge with largest cost. Hence, if $c_{kl} < c_{ij}$ for some tree edge (i, j) on this cycle, removing it and replacing it with (k, l) decreases the cost of our spanning tree. This necessary condition can actually be shown to be sufficient as well (see Exercise 5.19).

Path Optimality Condition Given a network G , a spanning tree T^* is a minimum spanning tree if and only if, for every nontree edge (k, l) ,

$$(5.5) \quad c_{ij} \leq c_{kl}$$

for every edge (i, j) contained on the path in T^* connecting nodes k and l .

Generating Constructive Algorithms: Prim's and Kruskal's Algorithms

We've already seen that we can derive a basic local search algorithm for the minimum spanning tree problem by simply using either of the above optimality conditions. Can we use these conditions to generate a constructive algorithm as well? This is not possible for every optimization problem, but it is for the minimum spanning tree problem. Any such algorithm must add edges one at a time, ensuring that the optimality conditions will be satisfied once the entire tree has been constructed.

Let's first consider the cut optimality condition. It may not seem obvious, however, that by adding edges one at a time so that cut optimality conditions are satisfied for a given cut, the cut optimality condition holds for all cuts. This is, in fact, the case.

Let T^0 be a subset of the edges of some minimum spanning tree, and suppose S is a set of nodes of some component of T^0 and $\bar{S} = V - S$. If (i, j) is a minimum cost edge in the cut (S, \bar{S}) , there exists a minimum spanning tree T^* that contains all edges in T^0 and the edge (i, j) .

The proof is left as an exercise. This implies that if we start with no edges and add edges that are the minimum cost edges in some cut, we can iteratively find a minimum spanning tree containing these edges. We can maintain a set S of nodes that have already been considered and are “attached” to the tree, and at each step add another node from \bar{S} to the tree using (5.4). This approach is often referred to as **Prim's algorithm**. To do this, we keep a tree that spans S and add a minimum cost neighbor to S by identifying an edge (i, j) in the cut (S, \bar{S}) . This edge is added to our tree, node j is added to S , and we proceed until all nodes are in our tree. A complete description of the algorithm is given in Algorithm 5.6.

Note that the correctness of Prim's algorithm follows directly from the cut

optimality condition since for each cut we have chosen the minimum cost edge to add to our tree.

Algorithm 5.6 Prim's Algorithm

$S = \{1\}$	$\{S \text{ contains the nodes currently in the tree}\}$
$T = \emptyset$	$\{T \text{ contains the current tree edges}\}$
while $S \neq V$ do	
Find an edge (i, j) of minimum cost where $i \in S, j \notin S$	
$T \leftarrow T \cup \{(i, j)\}, S \leftarrow S \cup \{j\}.$	
end while	

■ EXAMPLE 5.13

Let's illustrate Prim's algorithm on the network given in Example 5.12. If we start with $S = \{1\}$, we'd consider which of the edges $(1, 2)$ and $(1, 3)$ to add. Since $c_{12} = 4 < c_{13} = 6$, we add $(1, 2)$ to T and 2 to S , giving $S = \{1, 2\}$ and $T = \{(1, 2)\}$. We next consider the edges $(1, 3), (2, 3), (2, 4), (2, 5)$ that span the cut $(\{1, 2\}, \{3, 4, 5, 6\})$. Since edge $(2, 5)$ has the minimum cost, we add it to T and 5 to S , giving $S = \{1, 2, 5\}$ and $T = \{(1, 2), (2, 5)\}$. In the next three iterations, we'd add the edges (in order) $(4, 5), (3, 5)$, and $(4, 6)$, giving the minimum spanning tree $T = \{(1, 2), (2, 5), (4, 5), (3, 5), (4, 6)\}$.

Another idea that sounds reasonable for solving the minimum spanning tree problem (and maybe others as well) is, at each iteration, to simply choose from the remaining edges the edge (i, j) with minimum cost. If this edge, when added to our current set of edges, does not form a cycle, we keep it; if it forms a cycle, we discard it. In fact, for minimum spanning trees, this approach does work. Why? Suppose that an edge (i, j) is not kept because it forms a cycle. Since we have chosen the edges in nondecreasing order of their costs, all other edges on the cycle have cost no larger than that of (i, j) . Hence, by the path optimality (5.5), (i, j) cannot be in this minimum spanning tree if these other edges are there. This approach, known as **Kruskal's algorithm**, is given formally in Algorithm 5.7.

Algorithm 5.7 Kruskal's Algorithm

Sort the edges in nondecreasing order of costs c_{ij} .

$T = \emptyset$	$\{ \text{Tree edges} \}$
$S_i = \{i\}$	$\{ \text{Individual connected components of the tree} \}$
while $ T < (n - 1)$ do	
Let (i, j) be next edge from list of sorted edges to be examined.	
if i, j are both in the same component then	
Do Nothing	
else	
Let S_i and S_j denote the components containing nodes i, j , respectively.	
Merge S_i and S_j into the same component.	
$T \leftarrow T \cup (i, j)$	
end if	
end while	

■ EXAMPLE 5.14

Let's again use the network from Example 5.12. Ordering the edges by their cost, we have $\{(2, 5), (4, 5), (2, 3), (3, 5), (2, 4), (4, 6), (1, 2), (1, 3), (5, 6)\}$. We start with components $S_1 = \{i\}$. Looking at edge $(2, 5)$ first, since they are in separate components, we merge them, giving $S_2 = \{2, 5\}$, $S_5 = \emptyset$, and we add $(2, 5)$ to T . Next, since nodes 4, and 5 are in separate components S_4 and S_2 , we combine these components to form $S_2 = \{2, 4, 5\}$ and $S_4 = \emptyset$ and add $(4, 5)$ to our tree. Same holds for $(2, 3)$; however, at this point $(3, 5)$ is not added to T because 3 and 5 are in the same component, and adding this edge would yield the cycle $(2-3-5-2)$. Proceeding, we would add the edges $(4, 6)$ and $(1, 2)$ to our tree. Hence, our final tree would consist of the edges $\{(2, 5), (4, 5), (2, 3), (4, 6), (1, 2)\}$.

As you may have noticed, much more work is done when optimality conditions are derived and algorithms are designed to exploit them. However, this design approach is an important one that we will later exploit when designing algorithms to solve linear programs.

Summary

In this chapter, we explored the basics of algorithm design by constructing algorithms for various discrete optimization problems. We discussed the basic elements of constructive and local search methods, which are going to be prevalent in the forthcoming chapters. As we noted earlier, being able to design algorithms (heuristic or exact) for an optimization problem is one of

the most useful techniques for operations researchers since many of the problems we encounter require some new algorithmic approach to obtain a solution in a reasonable amount of time. In the next three chapters, we will use these basic constructs and employ them with a focus on solving linear programs. We will derive an algorithm from first principles, answering the basic questions mentioned in Section 5.2 that are to be asked when designing an algorithm. By Chapter 8, we should have a fair idea of the thinking involved when designing our own algorithms.

EXERCISES

5.1 Use the greedy heuristic to solve the following 0–1 knapsack problem

$$\max \quad 18x_1 + 15x_2 + 35x_3 + 25x_4 + 19x_5 + 10x_6 + 40x_7 + 30x_8$$

s.t.

$$6x_1 + 4x_2 + 7x_3 + 4x_4 + 5x_5 + 3x_6 + 10x_7 + 8x_8 \leq 30$$

$$x_i \in \{0, 1\}, \quad i \in \{1, 2, \dots, 8\}.$$

5.2 Consider the following alternative form of the 0–1 knapsack problem:

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

s.t.

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

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

Design a neighborhood search algorithm for this problem. Implement this algorithm in the following problem:

$$\min 6x_1 + 4x_2 + 7x_3 + 4x_4 + 5x_5 + 3x_6 + 10x_7 + 8x_8$$

s.t.

$$18x_1 + 15x_2 + 35x_3 + 25x_4 + 19x_5 + 10x_6 + 40x_7 + 30x_8 \geq 116$$

$$x_i \in \{0, 1\}, \quad i \in \{1, \dots, 8\}.$$

5.3 Consider the 0–1 knapsack problem

$$\begin{aligned}
\max \quad & \sum_{j=1}^n c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^n a_j x_j \leq b \\
& x_j \in \{0, 1\}, \quad j \in \{1, 2, \dots, n\},
\end{aligned}$$

where

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \dots \geq \frac{c_n}{a_n}.$$

Let z^*_{LP} be the optimal value of the continuous knapsack problem and let z_H denote the greedy heuristic solution to the corresponding 0–1 knapsack problem.

(a) Show that $z^*_{LP} \leq 2z_H$.

(b) Show that $z^*_{LP} \leq z_H + c_{\max}$, where $c_{\max} = \max\{c_j : j = 1, 2, \dots, n\}$.

5.4 Solve the following minimum cardinality set covering problem using the greedy heuristic.

$$\begin{aligned}
\min \quad & x_1 + x_2 + \dots + x_{12} \\
\text{s.t.} \quad & x_1 + x_3 + x_5 + x_7 + x_8 + x_9 \geq 1 \\
& x_2 + x_8 + x_9 \geq 1 \\
& x_1 + x_4 + x_7 + x_8 + x_9 \geq 1 \\
& x_4 + x_{10} \geq 1 \\
& x_1 + x_4 + x_6 \geq 1 \\
& x_6 + x_{10} + x_{11} \geq 1 \\
& x_1 + x_3 + x_5 + x_7 + x_{12} \geq 1 \\
& x_1 + x_2 + x_3 + x_8 + x_9 \geq 1 \\
& x_4 + x_6 + x_{10} + x_{11} + x_{12} \geq 1 \\
& x_i \in \{0, 1\}, \quad i \in \{1, 2, \dots, 12\}.
\end{aligned}$$

5.5 Suppose that we wanted to weight each variable in the set covering

problem. For example, suppose that the objective function in Problem 5.4 was

$$\begin{aligned} \min & 5x_1 + 2x_2 + 6x_3 + x_4 + 2x_5 + 4x_6 + 3x_7 + 2x_8 + 3x_9 \\ & + 5x_{10} + x_{11} + x_{12}. \end{aligned}$$

Modify the greedy heuristic for set covering problems to handle such objective functions. Use your algorithm to solve the set covering problem with the above objective function and the same constraints as in Problem 5.4.

5.6 Given a set covering problem, show that if there exist two sets S_j and S_k where $S_j \subseteq S_k$, then the constraint

$$\sum_{i \in S_k} \geq 1$$

can be eliminated from the problem without a change in the feasible region.

5.7 Set Packing Problem. A problem similar to the set covering problem is the set packing problem. Here, our optimization model is

$$\begin{aligned} \max & c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\ \text{s.t.} & \\ & \sum_{j \in S_i} x_j \leq 1, \quad i \in \{1, 2, \dots, m\} \\ & x_j \in \{0, 1\}, \quad j \in \{1, 2, \dots, m\}, \end{aligned}$$

where $S_i \subset \{1, 2, \dots, n\}$. Derive a greedy heuristic for the maximum cardinality set packing problem (where each $c_j = 1$) and use it to obtain a solution to the following problem:

$$\begin{aligned}
\max \quad & x_1 + x_2 + \cdots + x_{12} \\
\text{s.t.} \quad & x_1 + x_3 + x_5 + x_7 + x_8 + x_9 \leq 1 \\
& x_2 + x_8 + x_9 \leq 1 \\
& x_1 + x_4 + x_7 + x_8 + x_9 \leq 1 \\
& x_4 + x_{10} \leq 1 \\
& x_1 + x_4 + x_6 \leq 1 \\
& x_6 + x_{10} + x_{11} \leq 1 \\
& x_1 + x_3 + x_5 + x_7 + x_{12} \leq 1 \\
& x_1 + x_2 + x_3 + x_8 + x_9 \leq 1 \\
& x_4 + x_6 + x_{10} + x_{11} + x_{12} \leq 1 \\
& x_i \in \{0, 1\}, \quad j \in \{1, \dots, 12\}.
\end{aligned}$$

5.8 Consider the set packing problem introduced in Exercise 5.7. Derive a greedy algorithm for obtaining a solution for the general case (any c_j). Use it to find a solution to the problem where the constraints are the same as in Exercise 5.7 and the objective function is

$$\begin{aligned}
\max \quad & 5x_1 + 2x_2 + 6x_3 + x_4 + 2x_5 + 4x_6 + 3x_7 + 2x_8 + 3x_9 \\
& + 5x_{10} + x_{11} + x_{12}.
\end{aligned}$$

5.9 In the manufacturing of printed circuit boards, holes need to be drilled on the boards through which chips and other components are later wired. It is required that these holes be drilled as quickly as possible; hence, the problem of the most efficient order to drill the holes is a traveling salesman problem. Given the table below are the distances (in millimeters) between any pair of hole locations. Hence, minimum time equals finding the minimum total distance traveled between the hole locations.

Hole Locations	1	2	3	4	5	6	7	8
1	—	13	8	15	7	16	19	21
2	13	—	5	7	14	22	11	14
3	8	5	—	15	17	9	13	12
4	15	7	15	—	8	7	9	10
5	7	14	17	8	—	12	18	11
6	16	22	9	7	12	—	8	14
7	19	11	13	9	18	8	—	15
8	21	14	12	10	11	14	15	—

(a) Use the nearest neighbor heuristic to find an efficient ordering of the holes.

(b) Use the cheapest insertion heuristic to find an efficient ordering of the holes.

(c) For each solution found in parts (a) and (b), use an interchange algorithm to improve upon the orderings.

(d) Solve the TSP to optimality using an IP model discussed in Chapter 3. How many subtour elimination constraints are needed?

5.10 Show that the solution obtained in Example 5.11 after performing an interchange is the optimal solution to the problem.

5.11 In Exercise 5.9, there was only one drilling machine used to drill all the holes. Suppose that there were two machines that could drill holes simultaneously. Derive a heuristic to determine which machine drills which holes. Assume that there are n holes that need to be drilled, a distance d_{ij} between any two hole locations, and that each machine must end at the same hole location at which it begins.

5.12 A local beverage supplier has a depot at location 1 from where they must deliver vending supplies to customers at locations 2,...,n. Each customer has a demand for d_j units, and the distance between locations i and j is given by c_{ij} . If each truck can carry at most s units, develop an algorithm to minimize the total distance traveled so as to meet all demands. Apply this algorithm to the following example, where each truck can carry at most 120 units.

		Distances (miles)							
		Location							
		1	2	3	4	5	6	7	Deliver
Location 1	—	30	20	10	10	20	15		
2	30	—	10	20	40	50	35	50	
3	20	10	—	30	10	40	20	75	
4	10	20	30	—	15	25	20	80	
5	10	40	10	15	—	30	25	40	
6	20	50	40	25	30	—	10	30	
7	15	35	20	20	25	10	—	60	

5.13 In Exercise 5.12, now suppose that each truck can travel a distance of not more than L . Modify your algorithm to include this constraint and apply it to the given example where each truck can travel not more than 75 miles.

5.14 Recall from Chapter 2 that the transportation problem is one where m supply locations with supplies $s_i, i = 1, \dots, m$, are sending their supplies to n demand locations with demands $d_j, j = 1, \dots, n$. Associated with each shipment of material from supply location i to demand location j is a cost c_{ij} per unit. The goal is to meet all demands at minimum cost such that each supply location does not send out more material than it has available. Develop a heuristic for this problem and use it to find a solution for the following problem instance:

Supply Locations	Demand Locations				Capacities
	1	2	3	4	
1	20	45	35	10	50
2	35	35	50	20	30
3	30	20	15	25	20
Demands	30	25	40	5	

5.15 Single Machine Weighted Completion Time Sequencing. Suppose that there are n jobs to be processed by a single machine. All the jobs are available for processing at time $t = 0$, and the machine can process only one job at a time without interruption. For each job j , its processing time p_j and an associated weight w_j are given. Let C_j denote the completion time of job j . For example, suppose we are given the following information:

Jobs	1	2	3	4	5	6	7
w_j	0	18	12	8	8	17	16
p_j	3	6	6	5	4	8	9

If the jobs are sequenced in the order 1–3–5–7–2–4–6, then the completion times would be

Jobs	1	3	5	7	2	4	6
Completion times	3	9	13	22	28	33	41

If the objective is to minimize the total weighted completion time, that is,

$$\min \sum_{j=1}^n w_j C_j,$$

derivea heuristic to obtain a solution to this problem. Use this algorithm to find a sequencing of the jobs for the problem instance given above.

5.16 Single Machine Tardiness Sequencing. Suppose there are n jobs to be processed by a single machine. All jobs are available for processing at time $t = 0$, and the machine can process only one job at a time without interruption. For each job j , its processing time p_j and its due date d_j are given. Given an ordering of the jobs to be processed, the completion time C_j can be obtained for each job. In addition, the tardiness $T_j = \max\{C_j - d_j, 0\}$ of each job can be computed. Suppose that the objective is to find the sequence of jobs that minimizes the total tardiness $\sum_{j=1}^n T_j$. Develop a heuristic for this problem, and apply your algorithm to the following problem instance:

Job	Process Time	Due Date
1	33	35
2	17	90
3	6	43
4	40	85
5	22	60
6	8	23
7	27	76
8	19	59
9	30	99

5.17 Bin Packing Problem. Suppose you wish to pack n items of length a_i into bins of length L using the minimum number of such bins. Derive an algorithm to solve this problem and use it to obtain a solution for the following problem instance, where each bin is of length 12:

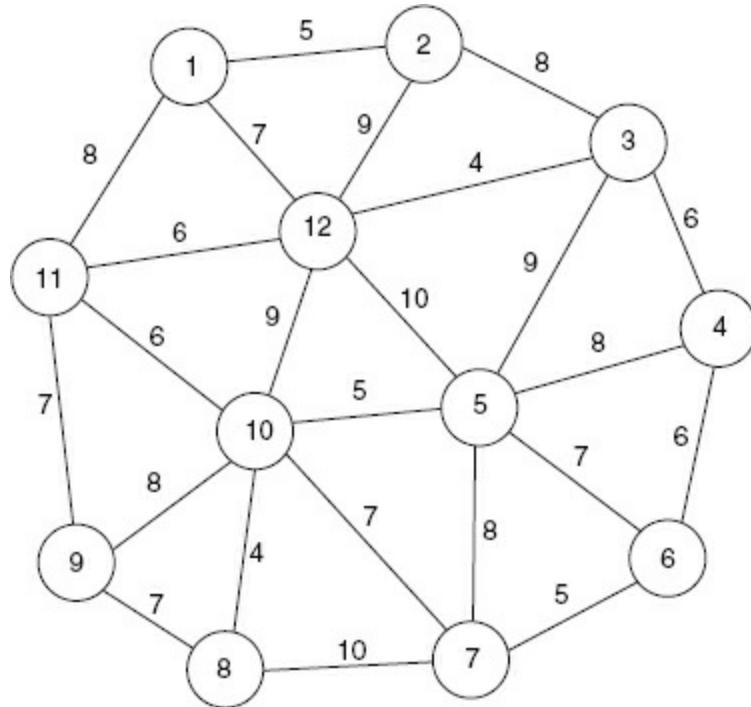
Item	1	2	3	4	5	6	7	8
Length	5	8	7	10	4	6	9	5

5.18 Given the graph in [Figure 5.12](#), solve the minimum spanning tree problem using

(a) Prim's algorithm.

(b) Kruskal's algorithm.

FIGURE 5.12 Graph for Exercise 5.18.



5.19 Prove the path optimality condition for minimum spanning trees: Given a network G , a spanning tree T^* is a minimum spanning tree if and only if, for every nontree edge (k, l) , $c_{ij} \leq c_{kl}$ for every edge (i, j) contained on the path in T^* connecting nodes k and l .

5.20 Let T^0 be a subset of the edges of some minimum spanning tree, and suppose S is a set of nodes of some component of T^0 and $S = V - S$. If (i, j) is a minimum cost edge in the cut (S, S) , prove that there exists a minimum spanning tree T^* that contains all edges in T^0 and the edge (i, j) .

CHAPTER 6

IMPROVING SEARCH ALGORITHMS AND CONVEXITY

Previous examples have shown that linear programming is useful in various real-world situations. Of course, many nonlinear optimization models are also useful, but none of these models would be practical if we could not actually solve these mathematical problems consistently.

In Chapter 1, we solved optimization problems on two variables using graphical techniques. This approach is of little importance for larger problems, mainly because we cannot “see” beyond three dimensions. However, it is insightful because it introduces several key concepts. In Chapter 5, we introduced the algorithm design process, including what we should consider when constructing an optimization algorithm. In this chapter, we review some useful techniques for solving optimization problems in multiple dimensions. In particular, we highlight some properties of linear programs that make them easy to solve, giving us the “building blocks” for constructing exact algorithms to solve linear programs.

We begin by discussing the basics of solving certain optimization problems. First, we give some definitions related to optimization, including what an optimal solution is. We continue with questions that should be addressed whenever we attempt to solve an optimization problem, and some of which we’ve already seen. We then discuss the concept of convexity, which enables us to identify if a given solution is in fact optimal. Finally, we mention a powerful result, known as Farkas’ lemma, that relates the optimality of linear programs to systems of equations and inequalities.

6.1 IMPROVING SEARCH AND

OPTIMAL SOLUTIONS

Many basic ideas in optimization can be neatly described by the following story. A blind man is left on a deserted island. He figures that if he can find the highest point on the island, then how would have the greatest chance of being spotted and rescued. So, how does he find this spot? Since he cannot see, he gets on his hands and knees and explores nearby, wanting to move to the highest point nearest to his current location. If repeated, eventually he'll reach a summit, right? But is this summit the highest point on the island? Unfortunately, probably not.

This story illustrates a “myopic” search for the best location. We look in our immediate “neighborhood” for a better locale, and move there, stopping only when our neighborhood consists of lower places. This is how a mathematical optimization algorithm generally works. The reason is that optimization algorithms are often algebraically myopic, in that they cannot see very far from its current location. Of course, there's more rigor than what we've described so far, but the idea is the same. This paradigm is often referred to as an *Improving Search Algorithm*, given in Algorithm 6.1, and many optimization algorithms use this concept. In contrast to solving linear programs geometrically where we moved whole lines, **this idea works from a single solution and tries to find another one with a better value.**

Algorithm 6.1 General Optimization Algorithm

1: Find an initial feasible solution \mathbf{x}^0 .
2: Set $k = 0$.
3: while \mathbf{x}^k is not optimal do
4: Determine a new solution \mathbf{x}^{k+1} that improves the objective value at \mathbf{x}^k .
5: Set $k = k + 1$.
6: end while

Notice that Algorithm 6.1 generates a sequence of solutions $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \dots$ that may be finite or infinite. Obviously, there are some questions regarding this very general algorithm.

1. What does it mean to be “optimal”?
2. How do we find a “better” solution \mathbf{x}^{k+1} ?

3. Does this algorithm actually stop (or at the very least converge) to a fixed solution?

We begin with the notion of optimal solution that we first saw in calculus.

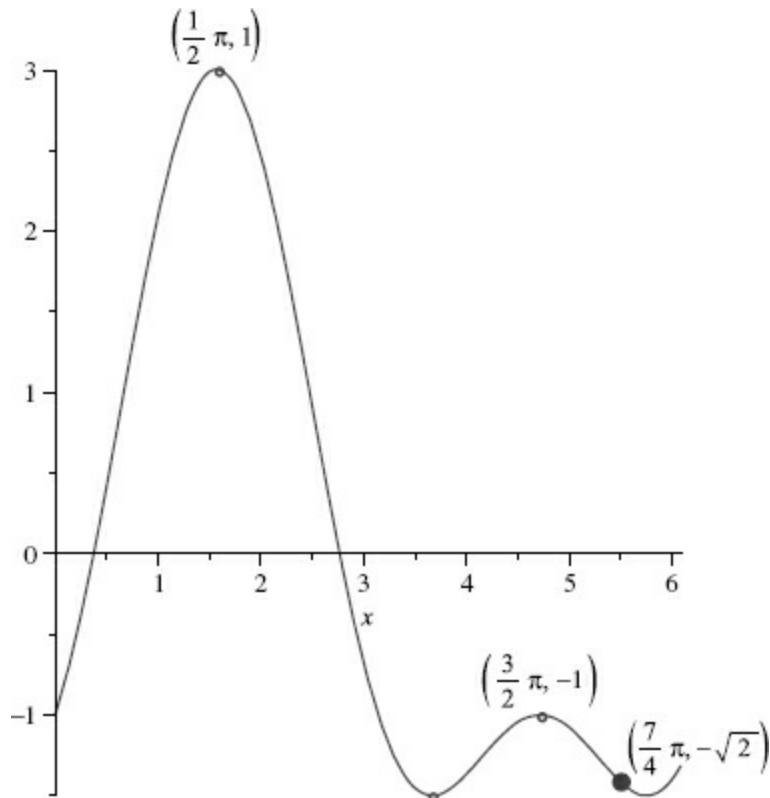
■ EXAMPLE 6.1

Consider the one-variable problem of finding the maximum of

$$f(x) = 2 \sin x - \cos 2x$$

over the interval $[0, 2\pi]$. The graph is given in [Figure 6.1](#). Suppose our current solution is $x^k = \frac{7\pi}{4}$. Since the derivative of $f(x)$ is $f'(x) = 2 \cos x + 2 \sin 2x$, and $f'(\frac{7\pi}{4}) < 0$, we know that by decreasing x a little from $\frac{7\pi}{4}$, our function value increases. Once we get to $x = \frac{3\pi}{2}$, any small movement from this value decreases our function value. But the maximum of this function occurs at $x = \frac{\pi}{2}$; in other words, $x = \frac{3\pi}{2}$ is “locally” optimal but not “globally” optimal. The algebraic myopia caused us to stop at $\frac{3\pi}{2}$ instead of heading to $x = \frac{\pi}{2}$.

FIGURE 6.1 Graph of $f(x) = 2 \sin x - \cos 2x$.



We need to make these notions of optimality more concrete before we can

proceed. Similar to our discussion in Chapter 5, “local” optimality is based upon a neighborhood.

ε -neighborhood The ε -neighborhood of a solution $\mathbf{x} \in \mathbb{R}^n$, denoted $N_\varepsilon(\mathbf{x})$, is the set of all solutions $\mathbf{y} \in \mathbb{R}^n$ whose distance from \mathbf{x} is less than ε , that is,

$$N_\varepsilon(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\| < \varepsilon\},$$

where $\|\mathbf{x}\|$ is the norm of the vector \mathbf{x} and is calculated as

$$\|\mathbf{x}\| = \sqrt{\sum_{j=1}^n x_j^2}.$$

Now that we have a notion of being “close” to a solution, we are ready to define what optimal means.

Local Optimal Solutions A solution \mathbf{x} to a continuous optimization problem is a *local optimum* if it is feasible and if there is an ε -neighborhood $N_\varepsilon(\mathbf{x})$ such that if $\mathbf{y} \in N_\varepsilon(\mathbf{x})$ and \mathbf{y} is feasible, then the objective value at \mathbf{y} is no better than the objective value at \mathbf{x} . If our problem is a maximization problem, then the value at \mathbf{y} is less than or equal to that at \mathbf{x} (and \mathbf{x} is called a **local maximum**), and the value at \mathbf{y} is greater than or equal to that at \mathbf{x} if our problem is a minimization problem (and \mathbf{x} is called a **local minimum**).

Global Optimal Solutions A solution \mathbf{x} to an optimization problem is a *global optimum* if it is feasible and if every other feasible solution \mathbf{y} has objective function value no better than that of \mathbf{x} .

In our example, $x = \frac{3\pi}{2}$ is a local maximum because if we set $\varepsilon = \frac{\pi}{8}$, for example, every solution in the interval $(\frac{3\pi}{2} - \frac{\pi}{8}, \frac{3\pi}{2} + \frac{\pi}{8})$ has a smaller function value than $\frac{3\pi}{2}$ (note that the local minima occur at $\frac{7\pi}{6}$ and $\frac{11\pi}{6}$). Similarly, $x = \frac{\pi}{2}$ is both a local and a global maximum although we generally only refer to it as a global maximum.

This example highlights how a general optimization algorithm typically works. We start at a feasible solution, determine how to improve it (based

upon our knowledge of the function and the feasible region), and continue until we cannot improve. However, as we've seen, there is a small problem.

A general optimization algorithm will often either stop if it reaches a local optimum, even if it is not the global optimum, or converge to a local optimum.

Unfortunately, in most situations, we must resign ourselves to this fate and live with local optima, with no idea if they are globally optimal. There are classes of optimization problems (and linear programming is one of these) where an optimization algorithm can be designed to always find a global optimum (if one exists). We'll see why later. But first, we discuss how to move from one solution to an improving one.

6.2 FINDING BETTER SOLUTIONS

When designing an optimization algorithm, the fundamental question is “how do we find a better solution?” Much like the story of the blind man trying to find the top of the hill, we find a direction for which we believe our objective function improves and then take a positive step in that direction. Mathematically, we state this as

$$(6.1) \quad \mathbf{x}^{k+1} = \mathbf{x}^k + \lambda \mathbf{d},$$

where \mathbf{x}^k is our current solution, \mathbf{d} is a direction vector, and $\lambda > 0$ is a scalar that indicates how far we move. Since we want \mathbf{x}^{k+1} to have a better value than \mathbf{x}^k , we want our direction \mathbf{d} to be “improving.”

Improving Direction A vector $\mathbf{d} \neq \mathbf{0}$ is an *improving direction* at the current solution \mathbf{x}^k if there exists a $\hat{\lambda} > 0$ such that the objective function value of $\mathbf{x}^k + \lambda \mathbf{d}$ is better than that of \mathbf{x}^k for all $0 < \lambda \leq \hat{\lambda}$.

■ EXAMPLE 6.2

In Example 6.1, if $x^k = \frac{7\pi}{4}$, then $d = -1$ is an improving direction and one

possible value of $\hat{\lambda}$ is $\frac{7\pi}{4} - \frac{3\pi}{2} = \frac{\pi}{4}$

How do we find such an improving direction, and will it always lead us toward an optimal solution? For continuous optimization problems, the improving direction is associated with the derivative.

Recall that for a one-variable (differentiable) function $f(x)$, the derivative $f'(x)$ measures the rate of change of f at x . As we saw in Example 6.1, if $f'(x) > 0$, then our function is increasing at x , meaning that increasing x by a small amount increases the function value. Similarly, decreasing x decreases the function value. If $f'(x) < 0$, then our function is decreasing at x , and increasing x decreases the function value and decreasing x increases the function value. These give us “improving directions” at x .

When we have a multivariable function $f(x_1, x_2, \dots, x_n) = f(\mathbf{x})$, we generalize this idea by using the *gradient* of $f(\mathbf{x})$,

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix},$$

where $\frac{\partial f}{\partial x}$ is the *partial derivative* of f with respect to variable x . As we saw in calculus, gradients have the following nice properties.

Given the problem of **maximizing** a differentiable function $f(\mathbf{x})$

- (a) the gradient $\nabla f(\mathbf{x})$ is an improving direction.
- (b) a vector \mathbf{d} is an improving direction if it makes an acute angle with the gradient; that is,

$$(6.2) \quad \nabla f(\mathbf{x})^T \mathbf{d} > 0.$$

Given the problem of **minimizing** a differentiable function $f(\mathbf{x})$

- (a) the negative of the gradient $-\nabla f(\mathbf{x})$ is an improving direction.
- (b) a vector \mathbf{d} is an improving direction if it makes an obtuse angle with the gradient; that is,

$$(6.3) \quad \nabla f(\mathbf{x})^T \mathbf{d} < 0.$$

These results tell us whether a direction is improving, but they do not tell us (except for the gradient itself) how to find an improving direction. In fact, the determination of an improving direction is one of the constructs that makes any two optimization algorithms different. Most algorithms use different criteria for selecting an improving direction, giving us tremendous flexibility when designing algorithms.

■ EXAMPLE 6.3

Consider the problem

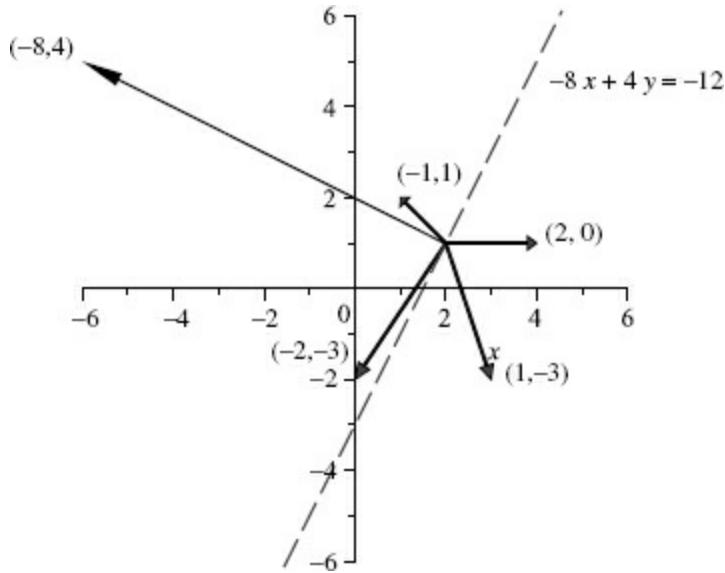
$$\max f(\mathbf{x}) = -x^3 + 4xy - 2y^2 + 1.$$

The gradient is

$$\nabla f(\mathbf{x}) = \begin{bmatrix} -3x^2 + 4y \\ 4x - 4y \end{bmatrix}.$$

Suppose that we are currently at the solution $(2, 1)$, so that $\nabla f(2, 1) = (-8, 4)$. It is easy to check that the directions $(-8, 4)$, $(-1, 1)$, and $(-2, -3)$ are all improving using (6.2), while the directions $(1, -3)$ and $(2, 0)$ are not improving. We can also verify this graphically by considering [Figure 6.2](#). Note that the line $-8x + 4y = -12$ is the unique line through $(2, 1)$ with normal vector $(-8, 4)$. Directions $(-8, 4)$, $(-1, 1)$, and $(-2, -3)$ all make acute angles with the gradient $\nabla f(2, 1) = (-8, 4)$ —they are all on the same side of the dashed line $-8x + 4y = -12$ as the gradient—while directions $(1, -3)$ and $(2, 0)$ make obtuse angles with the gradient.

FIGURE 6.2 Gradient and potential search directions from initial point $(2, 1)$.



Calculating Step Size

Recall the story of the blind man trying to find the top of the hill. He could move in a direction until he could no longer go higher moving in that direction; we shall take the same approach. Here, it seems natural to go from our current solution \mathbf{x}^k to a new solution in an improving direction \mathbf{d} by moving to $\mathbf{x}^{k+1} = \mathbf{x}^k + \hat{\lambda}\mathbf{d}$, for some $\hat{\lambda} > 0$. A common way to find $\hat{\lambda}$ is to find the largest value for which $f(\mathbf{x}^k + \lambda\mathbf{d})$ improves for $0 < \lambda \leq \hat{\lambda}$, where \mathbf{x}^k and \mathbf{d} are fixed vectors; in other words, create a one-variable optimization problem, which we know how to solve! We then use this value of $\hat{\lambda}$ to find our new solution $\mathbf{x}^{k+1} = \mathbf{x}^k + \hat{\lambda}\mathbf{d}$. This search approach is referred to as a **Line Search** or **Improving Search**, where the improving direction \mathbf{d} is often called a *search direction*, and the value of $\hat{\lambda}$ is referred to as the *step size*.

■ EXAMPLE 6.4

In Example 6.3, if $\mathbf{x}^k = (2, 1)$ and $\mathbf{d} = (-1, 1)$, our new solution will be $\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda\mathbf{d} = (2 - \lambda, 1 + \lambda)$, and we want the value of $\hat{\lambda} > 0$ that produces the first local maximum of

$$f(\mathbf{x}^k + \lambda\mathbf{d}) = g(\lambda) = -(2 - \lambda)^3 + 4(2 - \lambda)(1 + \lambda) - 2(1 + \lambda)^2 + 1.$$

Since this is a one-variable optimization problem, and we know that $g(\lambda)$ initially increases when λ increases from 0, we can simply find the first positive local optimal solution by evaluating the critical points of $g(\lambda)$ and

choosing the smallest positive value. Differentiating $g(\lambda)$ and simplifying gives

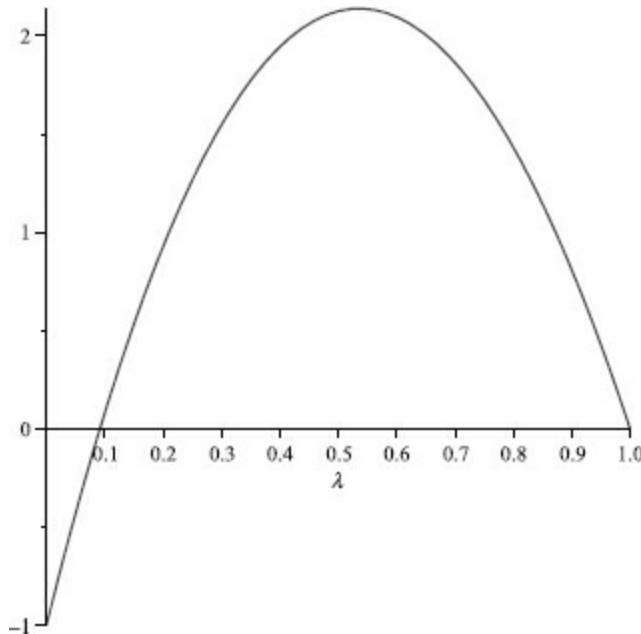
$$g' = 3\lambda^2 - 24\lambda + 12,$$

generating the smallest positive critical point value $\hat{\lambda} = 4 - \frac{1}{2}\sqrt{3} \approx 0.535$ ([Figure 6.3](#)). Note that we could choose a larger value of $\hat{\lambda}$ and still improve the function value (e.g., $\hat{\lambda} = 1$ satisfies $g(1) > g(0)$), but $\hat{\lambda} \approx 0.535$ is the first local maximum of $g(\lambda)$ found where $\lambda > 0$. Our new solution would then be $x^{k+1} = (2, 1) + 0.535(-1, 1) = (1.465, 1.535)$. Is this solution a local maximum of $f(\mathbf{x})$? To check, we'd need to compute

$$\nabla f(1.465, 1.535) = \nabla f(\mathbf{x}^{k+1}) \approx \begin{bmatrix} -0.299 \\ -0.280 \end{bmatrix}.$$

Since $\nabla f(x, y) \neq \mathbf{0}$, it is an improving direction, and so our new solution is not a local maximum. What would we do from here? We'd repeat our steps at this new solution until we found a solution (x, y) where $\nabla f(x, y) \approx \mathbf{0}$ (this is because when we do numerical calculations, we hardly ever get 0 for an answer. We often have to settle for values close to 0).

FIGURE 6.3 Graph of $g(\lambda)$ over $0 < \lambda < 1$.



Dealing with Constraints

We have ignored the possibility that our problem may be constrained.

Previously, we found our step size by solving a one-variable optimization problem. However, if we have constraints, we need to ensure they are satisfied.

■ EXAMPLE 6.5

Let's consider the linear program

$$\begin{aligned} \max \quad & f(x, y) = 13x + 5y \\ \text{s.t.} \quad & 4x + y \leq 24 \\ & x + 3y \leq 24 \\ & 3x + 2y \leq 23 \\ (6.4) \quad & x, y \geq 0. \end{aligned}$$

Since our objective function is linear, the gradient at any solution (feasible or not) is $\nabla f = (13, 5)$. If we start at the solution $(2, 1)$, and we use the gradient as our search direction, then our new solution is

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} + \lambda \begin{bmatrix} 13 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 + 13\lambda \\ 1 + 5\lambda \end{bmatrix},$$

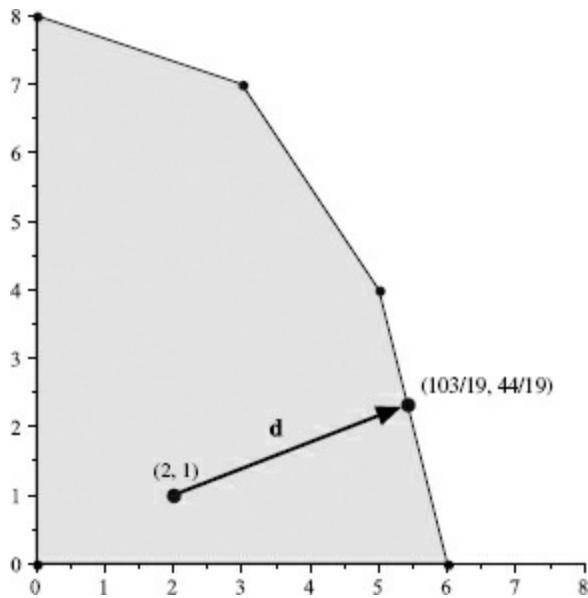
for some $\lambda \geq 0$. To find the step size, we maximize

$$\begin{aligned} g(\lambda) &= 13(2 + 13\lambda) + 5(1 + 5\lambda) \\ &= 31 + 194\lambda. \end{aligned}$$

Unfortunately, in our line search we continue to improve as $\lambda \rightarrow \infty$, but for large λ , we are no longer feasible. Note that for linear objectives this is **always true!** From [Figure 6.4](#), we see that moving from $(2, 1)$ in the direction $(13, 5)$ strikes

the constraint $4x + y \leq 24$. Since it makes sense to remain feasible, we need to find the value of λ for which the solution $(2, 1) + \lambda(13, 5) = (2 + 13\lambda, 1 + 5\lambda)$ satisfies the constraint $4x + y \leq 24$. Notice that $\hat{\lambda} = \frac{15}{57}$ solves $4(2 + 13\lambda) + (1 + 5\lambda) = 24$ and that, using this step size, our new solution is $(\frac{103}{19}, \frac{44}{19})$. Our objective function value has improved from 31 at $(2, 1)$ to $82\frac{1}{19}$ at $(\frac{103}{19}, \frac{44}{19})$.

[FIGURE 6.4](#) Improving direction $\mathbf{d} = (13, 5)$ from $(2, 1)$.



The same process is possible without visualizing the feasible region. Generally, starting at a solution \mathbf{x} , if the objective improves along an improving direction \mathbf{d} and we remain feasible for some range of step size λ , then we find the largest possible step size that maintains feasibility. So, for the problem in Example 6.5, we find a value of λ such that

$$\begin{aligned} 4(2 + 13\lambda) + (1 + 5\lambda) &\leq 24 \\ (2 + 13\lambda) + 3(1 + 5\lambda) &\leq 24 \\ 3(2 + 13\lambda) + 2(1 + 5\lambda) &\leq 23. \end{aligned}$$

These inequalities imply that

$$\lambda \leq \frac{15}{57}, \quad \lambda \leq \frac{19}{28}, \quad \text{and} \quad \lambda \leq \frac{15}{36}.$$

Our choice of the (maximum) step size $\hat{\lambda}$ satisfies **each** of these, and hence

$$\begin{aligned} \hat{\lambda} &= \min \left\{ \frac{15}{57}, \frac{19}{28}, \frac{15}{36} \right\} \\ &= \frac{15}{57}. \end{aligned}$$

Note that any positive step size smaller than $\hat{\lambda}$ gives a feasible solution with an improved objective value, so we need not always choose the maximum step size for our algorithm.

■ EXAMPLE 6.6

Let's again consider the linear program (6.4) and start at the solution $\mathbf{x}^k = (2, 1)$, but this time choose as our search direction $\mathbf{d} = (3, -1)$. Since

$$\nabla f(\mathbf{x}^k)^T \mathbf{d} = [13 \quad 5] \begin{bmatrix} 3 \\ -1 \end{bmatrix} > 0,$$

this is an improving search direction. We need to ensure that our new solution $\mathbf{x}^{k+1} = (2, 1) + \lambda(3, -1) = (2 + 3\lambda, 1 - \lambda)$ satisfies the constraints

$$4(2 + 3\lambda) + (1 - \lambda) \leq 24$$

$$(2 + 3\lambda) + 3(1 - \lambda) \leq 24$$

$$3(2 + 3\lambda) + 2(1 - \lambda) \leq 23,$$

yielding the bounds on λ

$$11\lambda \leq 15, \quad 0 \leq 19, \quad \text{and} \quad 7\lambda \leq 15.$$

Thus,

$$\hat{\lambda} = \min \left\{ \frac{15}{11}, \frac{15}{7} \right\} = \frac{15}{11},$$

and our new solution is

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} + \frac{15}{11} \begin{bmatrix} 3 \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{67}{11} \\ -\frac{4}{11} \end{bmatrix},$$

which is not feasible. Thus, in addition to choosing $\hat{\lambda}$ so that all constraints are satisfied, we must make sure that our new solution maintains its bounds. This means we also require

$$2 + 3\lambda \geq 0$$

$$1 - \lambda \geq 0,$$

which is the same as $\lambda \geq -\frac{2}{3}$ and $\lambda \leq 1$. Hence, our value $\hat{\lambda}$ is

$$\hat{\lambda} = \min \left\{ \frac{15}{11}, \frac{15}{7}, 1 \right\} = 1,$$

with our new solution being $(5, 0)$.

This improving search approach is not limited to linear programs, as the next example illustrates.

■ EXAMPLE 6.7

Consider the optimization problem

$$\begin{aligned}
\max \quad & f(x, y) = -x^2 + 4xy - 2y^2 \\
\text{s.t.} \quad & 4x + y \leq 24 \\
& x + 3y \leq 24 \\
& 3x + 2y \leq 23 \\
& x, y \geq 0.
\end{aligned}$$

The gradient vector of the objective function is

$$\nabla f = \begin{bmatrix} -2x + 4y \\ 4x - 4y \end{bmatrix}.$$

At the feasible solution $\mathbf{x} = (2, 1)$, the gradient is $\nabla f(2, 1) = (0, 4)$. If we move in the direction $\mathbf{d} = \nabla f(2, 1)$, the step size that generates a local maximum of

$$g(\lambda) = f(2 + 0\lambda, 1 + 4\lambda) = 4 + 32\lambda - 2(1 + 4\lambda)^2$$

is $\lambda = \frac{1}{4}$. To bound the step size using the constraints, we calculate

$$\begin{aligned}
4(2 + 0\lambda) + (1 + 4\lambda) \leq 24 &\implies \lambda \leq \frac{15}{4} \\
(2 + 0\lambda) + 3(1 + 4\lambda) \leq 24 &\implies \lambda \leq \frac{19}{12} \\
3(2 + 0\lambda) + 2(1 + 4\lambda) \leq 23 &\implies \lambda \leq \frac{15}{8}.
\end{aligned}$$

Thus, the largest step size that guarantees $\nabla f(2, 1) = (0, 4)$ is an improving direction and generates another feasible solution is $\hat{\lambda} = \frac{1}{4}$.

Alternatively, consider the direction $\mathbf{d} = (13, 5)$. This direction is improving since $\nabla f(2, 1)^T \mathbf{d} = 20 > 0$. We have already seen in the previous examples that the maximum step size we can take before hitting at least one of the constraints is $\hat{\lambda} = \frac{15}{57}$. To check the step size to ensure that \mathbf{d} is improving, note that

$$\begin{aligned}
g(\lambda) &= f(2 + 13\lambda, 1 + 5\lambda) \\
&= -2(2 + 13\lambda)^2 + 4(2 + 13\lambda)(1 + 5\lambda) - 2(1 + 5\lambda)^2 \\
&= 2 + 20\lambda + 41\lambda^2.
\end{aligned}$$

Since $g'(\lambda) = 20 + 82\lambda > 0$ for all $\lambda > 0$, $\mathbf{d} = (13, 5)$ is an improving direction for any step size. Thus, our maximum step size $\hat{\lambda} = \frac{15}{57}$ is determined by the constraints and not by the objective function.

The previous examples highlight the following important idea.

Rule In any constrained optimization problem, the step size $\hat{\lambda}$ needs to consider not only each constraint and each variable bound but also the objective function.

Finding Feasible Directions

So far, we've looked only at the case where the current solution does not satisfy any of the constraints at equality. When our solution does satisfy a constraint or bound at equality, more care is needed in selecting our improving direction.

■ EXAMPLE 6.8

Suppose we start at $(\frac{103}{19}, \frac{44}{19})$ in Example 6.5. If we choose as our search direction $(13, 5)$, we immediately find that $\hat{\lambda} = 0$ since we are already on one of the constraints.

However, being on any one constraint is not necessarily the problem. Suppose we start at the solution $(6, 0)$; this solution is on both the constraint $4x + y \leq 24$ and the variable bound $y \geq 0$. The vector $\mathbf{d} = (1, 0)$ is an improving direction. But, we would again find that our step size $\hat{\lambda} = 0$ because any positive step size in this direction violates the constraint $4x + y \leq 24$ even though the constraint $y \geq 0$ remains satisfied.

This example indicates that not all improving directions will work. Before we proceed, a few definitions will be useful.

Active Constraint Given a feasible solution \mathbf{x} to an optimization problem, a constraint satisfied at equality at \mathbf{x} is said to be an *active constraint* at \mathbf{x} . These constraints are also referred to as **tight** or **binding constraints**.

Feasible Direction A direction $\mathbf{d} \neq \mathbf{0}$ is a *feasible direction* at \mathbf{x} if there is some $\hat{\lambda} > 0$ such that $\mathbf{x} + \lambda\mathbf{d}$ is feasible for all $0 \leq \lambda \leq \hat{\lambda}$. It is possible that, along a feasible direction \mathbf{d} , $\mathbf{x} + \lambda\mathbf{d}$ is feasible for all $\lambda \geq 0$.

Special care is needed in the selection of an improving direction with regard to active constraints.

■ EXAMPLE 6.9

In Example 6.8, the constraint $4x + y \leq 24$ is active at $(\frac{103}{19}, \frac{44}{19})$, We find a search direction \mathbf{d} from this solution by guaranteeing that

$$\begin{bmatrix} \frac{103}{19} \\ \frac{44}{19} \end{bmatrix} + \lambda \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

is feasible for some $\lambda > 0$, which implies that

$$\begin{aligned} 4\left(\frac{103}{19} + \lambda d_x\right) + \left(\frac{44}{19} + \lambda d_y\right) &\leq 24 \\ 4\lambda d_x + \lambda d_y &\leq 0 \quad \text{or} \\ 4d_x + d_y &\leq 0. \end{aligned}$$

Because no other constraint is active, our search direction must be an improving direction that satisfies $4d_x + d_y \leq 0$. One example would be $\mathbf{d} = (-1, 4)$, since $\nabla f^T \mathbf{d} = 7 > 0$. Using this search direction, we then choose a step size so that none of the other constraints are violated. To find the step size $\hat{\lambda}$, we know that our next solution $(\frac{103}{19}, \frac{44}{19}) + \lambda(-1, 4)$ must satisfy

$$\begin{aligned} 4\left(\frac{103}{19} - \lambda\right) + \left(\frac{44}{19} + 4\lambda\right) &\leq 24 \\ \left(\frac{103}{19} - \lambda\right) + 3\left(\frac{44}{19} + 4\lambda\right) &\leq 24 \\ 3\left(\frac{103}{19} - \lambda\right) + 2\left(\frac{44}{19} + 4\lambda\right) &\leq 23 \\ \frac{103}{19} - \lambda &\geq 0 \\ \frac{44}{19} + 4\lambda &\geq 0, \end{aligned}$$

or equivalently,

$$\lambda \geq 0, \quad 11\lambda \leq \frac{221}{19}, \quad 5\lambda \leq \frac{40}{19}, \quad \text{and} \quad \lambda \leq \frac{103}{19}.$$

Our step size is

$$\hat{\lambda} = \min \left\{ \frac{221}{19(11)}, \frac{40}{19(5)}, \frac{103}{19} \right\} = \frac{40}{95} = \frac{8}{19}.$$

Note that due to the choice of \mathbf{d} and the fact that $\lambda > 0$, we satisfy the constraint $4x + y \leq 24$, so we need not consider it in the computation of the step size. If we move in the direction $(-1, 4)$ from $(\frac{103}{19}, \frac{44}{19})$ and take a step size $\hat{\lambda} = \frac{8}{19}$, our next solution is

$$\begin{bmatrix} \frac{103}{19} \\ \frac{44}{19} \end{bmatrix} + \frac{8}{19} \begin{bmatrix} -1 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}.$$

The objective value at $(5, 4)$ is 85, an improvement over the value of $82\frac{1}{19}$ at $(\frac{103}{19}, \frac{44}{19})$.

In this example, we had a linear constraint of the form $\mathbf{a}^T \mathbf{x} \leq b$ that was active, and so our search direction \mathbf{d} had to satisfy $\mathbf{a}^T \mathbf{d} \leq 0$. We can generalize this for any linear constraint.

A direction \mathbf{d} is a feasible direction at a feasible solution \mathbf{x} to a linearly constrained optimization problem if

- (a) $\mathbf{a}^T \mathbf{d} \leq 0$ for each active constraint of the form $\mathbf{a}^T \mathbf{x} \leq b$.
- (b) $\mathbf{a}^T \mathbf{d} \geq 0$ for each active constraint of the form $\mathbf{a}^T \mathbf{x} \geq b$.
- (c) $\mathbf{a}^T \mathbf{d} = 0$ for each active constraint of the form $\mathbf{a}^T \mathbf{x} = b$.

■ EXAMPLE 6.10

Consider the solution $(5, 4)$ in Example 6.9 that gives the active constraints of $4x + y \leq 24$ and $3x + 2y \leq 23$. Any feasible direction $\mathbf{d} = (d_x, d_y)$ at $(5, 4)$ must satisfy

$$\begin{aligned} 4d_x + d_y &\leq 0 \\ 3d_x + 2d_y &\leq 0. \end{aligned}$$

Detecting Unbounded Solutions

There is one more case we need to discuss.

■ EXAMPLE 6.11

Consider the following linear program

$$\begin{aligned} \max \quad & 7x + 5y \\ \text{s.t.} \quad & 7x + 2y \geq 28 \\ & 2x + 12y \geq 24 \\ & x, y \geq 0. \end{aligned}$$

[Figure 6.5](#) shows its feasible region and an objective function contour (with gradient vector), indicating that there is no finite optimal solution.

We can detect such a case using an improving search approach. Suppose

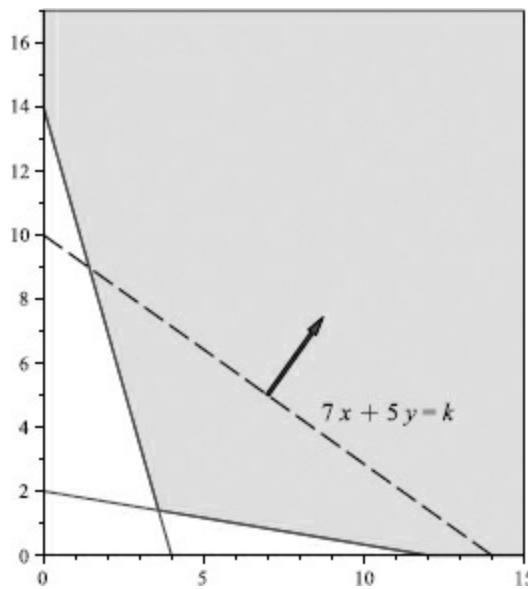
we're at the feasible solution $(4, 2)$ and our search direction is $(7, 5)$. In this case, we find our step size $\hat{\lambda}$ so that

$$7(4 + 7\lambda) + 2(2 + 5\lambda) \geq 28 \quad \text{and}$$

$$2(4 + 7\lambda) + 12(2 + 5\lambda) \geq 24.$$

It is easy to see that any $\lambda > 0$ satisfies both constraints, implying that there is no constraint to stop us from moving in this (improving) direction. Hence, this problem is unbounded.

FIGURE 6.5 Example of unbounded linear program.



Given a feasible solution \mathbf{x} to a linear program and an improving direction \mathbf{d} if $\mathbf{x} + \lambda\mathbf{d}$ is feasible for all $\lambda \geq 0$, then our optimization problem is unbounded.

General Improving Search Algorithm

We are now ready to summarize everything discussed so far. A general improving search algorithm is given in Algorithm 6.2. In the design of an improving search algorithm, we must decide how to find an initial starting solution, feasible search directions, and step lengths. In the next two chapters, we discuss how to do this for linear programs.

Algorithm 6.2. General Improving Search Algorithm

1: Find an initial feasible solution \mathbf{x}^0 . Set $k = 0$.

2: while \mathbf{x}^k is not a local optimum do
3: Pick an improving feasible direction \mathbf{d} .
4: If none exists, our current solution is a local optimum. STOP.
5: Given a search direction \mathbf{d} , choose step size $\hat{\lambda} > 0$ as the largest value such that \mathbf{d} continues to be an improving direction and feasibility is retained.
6: if $\hat{\lambda} \rightarrow \infty$ while still both improving and remaining feasible then
7: our problem is unbounded. STOP.
8: else
9: The new solution is $\mathbf{x}^{k+1} = \mathbf{x}^k + \hat{\lambda}\mathbf{d}$.
10: end if
11: Set $k = k + 1$.
12: end while

A natural question remains: Assuming our optimization problem is not unbounded, does an improving search algorithm stop or converge? This question depends not only on the improving feasible direction chosen but also on the step size (see Exercise 6.12). However, under some general conditions, we can ensure convergence. For details on these conditions, see the books by Nash and Sofer [66] and Bazarra, Sherali, and Shetty [9].

Under general conditions on the improving feasible direction and step size, Algorithm 6.2 converges to a local optimal solution if the optimization problem is not unbounded.

■ EXAMPLE 6.12

Let's continue with Algorithm 6.2 for the problem in Example 6.5. In Example 6.10, we found that the set of feasible directions at the solution (5, 4) was described as the solutions to the system of inequalities

$$4d_x + d_y \leq 0$$

$$3d_x + 2d_y \leq 0.$$

Do you notice that (5, 4) is a corner point of our feasible region, occurring at the intersection of two constraints? In fact, if we solve a linear program in two variables using Algorithm 6.2, where we always choose the improving direction making the smallest angle with the gradient vector and move the maximum step size allowed, eventually we will either show the problem to be

unbounded (as in Example 6.11) or we will reach a corner point.

Since the gradient of our objective function is $\nabla f = (13, 5)$, all improving directions satisfy

$$13d_x + 5d_y > 0.$$

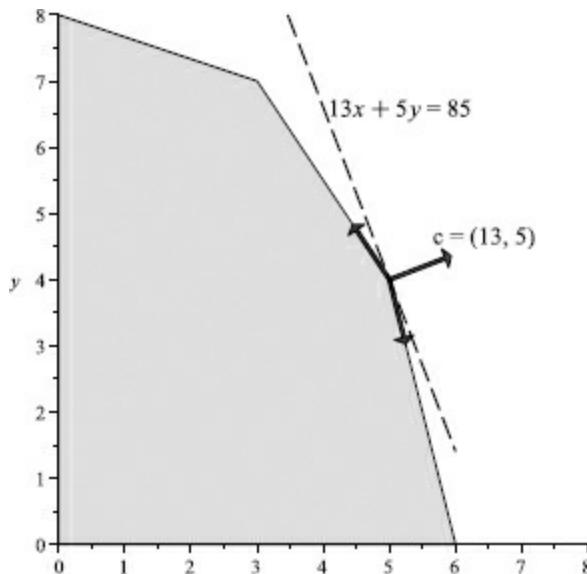
Are there any directions $\mathbf{d} = (d_x, d_y)$ satisfying all three inequalities? Consider [Figure 6.6](#). Note that all feasible directions at $(5, 4)$ point toward the inside of the feasible region or along the constraints

$$4x + y \leq 24$$

$$3x + 2y \leq 23;$$

the directions along the constraints are $\mathbf{d}^1 = (1, -4)$ and $\mathbf{d}^2 = (-2, 3)$. Since neither of these directions makes an acute angle with the gradient, we are convinced that no other feasible direction at $(5, 4)$ is improving. In Section 6.4 and in Chapter 7, we expand on this notion. Thus, the solution $(5, 4)$ must be a local maximum. Is it a global maximum? Algorithm 6.2 says nothing about this. What we need is additional mathematical analysis, which we'll begin to explore in the next section.

FIGURE 6.6 Feasible region for Example 6.12 showing $(5, 4)$ as local maximum.



6.3 CONVEXITY: WHEN DOES

IMPROVING SEARCH IMPLY GLOBAL OPTIMALITY?

Now that we've discussed the basics of an improving search algorithm, a natural question to ask is when does an improving search algorithm terminate with a global optimum? We know that an improving search algorithm generates a local optimum if it converges, but we cannot always guarantee global optimality. This section addresses a case when an improving search algorithm, if it converges, will always converge to a global optimum.

It seems reasonable that whatever conditions guarantee that a local optimum is also a global optimum, such conditions exist on both the objective function and the feasible region. Some classical conditions reduce to the study of how line segments relate.

Given two solutions \mathbf{x} , \mathbf{y} , the set of solutions on the line segment joining them can be written as

$$\{(1 - \lambda)\mathbf{x} + \lambda\mathbf{y} = \mathbf{x} + \lambda(\mathbf{y} - \mathbf{x}), 0 \leq \lambda \leq 1\}.$$

When $\lambda = 0$, our solution is \mathbf{x} , and when $\lambda = 1$, our solution is \mathbf{y} . Note that the line segment can also be written as

$$\{\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} = \mathbf{y} + \lambda(\mathbf{x} - \mathbf{y}), 0 \leq \lambda \leq 1\}.$$

■ EXAMPLE 6.13

Suppose $\mathbf{x} = (1, 3)$ and $\mathbf{y} = (5, -1)$. The line segment joining \mathbf{x} and \mathbf{y} is

$$\begin{aligned}(1 - \lambda)\mathbf{x} + \lambda\mathbf{y} &= (1 - \lambda) \begin{bmatrix} 1 \\ 3 \end{bmatrix} + \lambda \begin{bmatrix} 5 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 + 4\lambda \\ 3 - 4\lambda \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} + \lambda \begin{bmatrix} 4 \\ -4 \end{bmatrix},\end{aligned}$$

where $0 \leq \lambda \leq 1$. If, for example, we set $\lambda = \frac{3}{4}$ we would have the solution $(1 + 3, 3 - 3) = (4, 0)$, which is three-quarters of the distance from \mathbf{x} to \mathbf{y} along the line segment joining them. If λ was unrestricted, we would have the entire line containing these two solutions.

Convex Sets

Let's begin with a condition on the feasible region. Ideally, if we have solutions \mathbf{x}_1 and \mathbf{x}_2 with $f(\mathbf{x}_2)$ "better than" $f(\mathbf{x}_1)$, then we want to move from \mathbf{x}_1 in the direction of \mathbf{x}_2 and remain feasible.

FIGURE 6.7 Examples of convex sets.

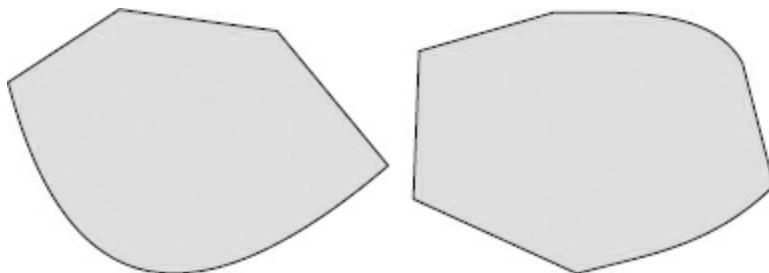
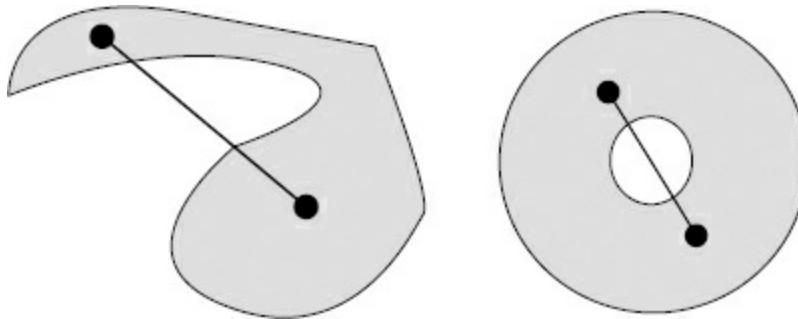


FIGURE 6.8 Examples of nonconvex sets.



Convex Set A set S is *convex* if for all $\mathbf{x}, \mathbf{y} \in S$, then $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in S$, for all $0 \leq \lambda \leq 1$.

In other words, a set is convex if, for any two solutions in the set, all solutions on the line segment joining these solutions are also in the set; see [Figure 6.7](#) for some sets that are convex and [Figure 6.8](#) for some that are not. In addition, we know a number of sets S that are convex. Verifying this fact algebraically, however, can be harder than initially thought.

■ EXAMPLE 6.14

Consider the set $S = \{(x, y) : x^2 + y^2 \leq 16\}$, as given in [Figure 6.9](#). To show that S is convex, let $(x_1, y_1), (x_2, y_2) \in S$. We show that, for $0 \leq \lambda \leq 1$,

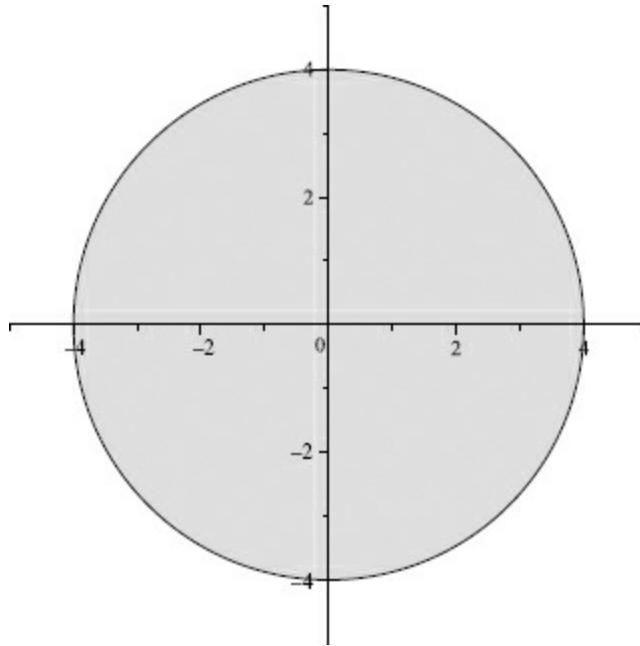
$$\lambda(x_1, y_1) + (1 - \lambda)(x_2, y_2) = (\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2) \in S.$$

To see this, note that

$$\begin{aligned}
& (\lambda x_1 + (1 - \lambda)x_2)^2 \\
& + (\lambda y_1 + (1 - \lambda)y_2)^2 = \lambda^2(x_1^2 + y_1^2) + (1 - \lambda)^2(x_2^2 + y_2^2) \\
& \quad + 2\lambda(1 - \lambda)(x_1x_2 + y_1y_2) \\
& \leq \lambda^2(16) + (1 - \lambda)^2(16) + 2\lambda(1 - \lambda)(16) \\
& = 16(\lambda + (1 - \lambda))^2 \\
& = 16,
\end{aligned}$$

because $(x_1x_2 + y_1y_2) = (x_1, y_1) \cdot (x_2, y_2) \leq \| (x_1, y_1) \| \| (x_2, y_2) \| = 4^2 = 16$. Hence, it follows that $(\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2) \in S$.

FIGURE 6.9 Set $S = \{(x, y) : x^2 + y^2 \leq 16\}$.



■ EXAMPLE 6.15

Consider the set $S = \{(x, y) : 3x + 2y \leq 12\}$. Let (x_1, y_1) and (x_2, y_2) be two solutions in S and consider the solution

$$\lambda(x_1, y_1) + (1 - \lambda)(x_2, y_2) = (\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2),$$

where $0 \leq \lambda \leq 1$. We see that

$$\begin{aligned}
& 3(\lambda x_1 + (1 - \lambda)x_2) \\
& + 2(\lambda y_1 + (1 - \lambda)y_2) = \lambda(3x_1 + 2y_1) + (1 - \lambda)(3x_2 + 2y_2) \\
& \leq \lambda(12) + (1 - \lambda)(12) \\
& = 12.
\end{aligned}$$

Thus, $\lambda(x_1, y_1) + (1 - \lambda)(x_2, y_2) \in S$, and so S is convex.

■ EXAMPLE 6.16

Let's show that the set $S = (x, y) : x^2 + y^2 \geq 16$ is not convex. Consider the solutions $(4, 0)$ and $(0, 4)$. Each solution is in S , but $\frac{1}{2}(4, 0) + \frac{1}{2}(0, 4) = (2, 2) \notin S$, and hence S is not convex.

We can generalize the definition of convexity to more than two points.

Convex Combination Let $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ be a set of solutions in \mathbb{R}^n .

Then \mathbf{x} is a *convex combination* of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ if

$$\mathbf{x} = \sum_{i=1}^k \alpha_i \mathbf{v}_i$$

for some values α_i satisfying

$$\begin{aligned} \sum_{i=1}^k \alpha_i &= 1 \\ \alpha_i &\geq 0, \quad i \in \{1, \dots, k\}. \end{aligned}$$

Note that if $k = 2$, then this implies that \mathbf{x} is on the line segment joining \mathbf{v}_1 and \mathbf{v}_2 . We can use this idea of convex combinations to form convex sets by knowing only a finite set of solutions.

Lemma 6.1 Let $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ be a set of solutions in \mathbb{R}^n . Then the set

$$S = \left\{ \mathbf{x} : \mathbf{x} = \sum_{i=1}^k \alpha_i \mathbf{v}_i, \sum_{i=1}^k \alpha_i = 1, \alpha_i \geq 0, \quad i \in \{1, \dots, k\} \right\}$$

is a convex set.

Proof Let $\mathbf{x}, \mathbf{y} \in S$. Then there exists α_i and β_i such that

$$\begin{aligned} \mathbf{x} &= \sum_{i=1}^k \alpha_i \mathbf{v}_i, \sum_{i=1}^k \alpha_i = 1, \quad \alpha_i \geq 0, \\ \mathbf{y} &= \sum_{i=1}^k \beta_i \mathbf{v}_i, \sum_{i=1}^k \beta_i = 1, \quad \beta_i \geq 0. \end{aligned}$$

We need to show that for any $\lambda \in [0, 1]$, $\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in S$.

Consider $\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}$ for some $\lambda \in [0, 1]$. Then

$$\begin{aligned}\lambda \mathbf{x} + (1 - \lambda) \mathbf{y} &= \lambda \sum_{i=1}^k \alpha_i \mathbf{v}_i + (1 - \lambda) \sum_{i=1}^k \beta_i \mathbf{v}_i \\ &= \sum_{i=1}^k (\lambda \alpha_i + (1 - \lambda) \beta_i) \mathbf{v}_i.\end{aligned}$$

If we can show that (i) $\sum_{i=1}^k (\lambda \alpha_i + (1 - \lambda) \beta_i) = 1$ and (ii) $\lambda \alpha_i + (1 - \lambda) \beta_i \geq 0$ for each i , then we are done. Point (ii) is obvious since $\lambda \in [0, 1]$ and $\alpha_i, \beta_i \geq 0$. To show (i), note that

$$\begin{aligned}\sum_{i=1}^k (\lambda \alpha_i + (1 - \lambda) \beta_i) &= \lambda \sum_{i=1}^k \alpha_i + (1 - \lambda) \sum_{i=1}^k \beta_i \\ &= \lambda(1) + (1 - \lambda)(1) \\ &= 1.\end{aligned}$$

Lemma 6.1 says that given any finite set of solutions, we can construct a convex set by just taking the convex combinations of those solutions. The set S generated is called the **convex hull** of $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ and is denoted by $\text{conv}(\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\})$.

■ EXAMPLE 6.17

Consider the set of solutions $P = \{(1, 2), (3, 4), (5, 3), (7, 0)\}$. The convex hull of P , or $\text{conv}(P)$, is shown in [Figure 6.10](#). Note that each solution in P is a corner point of $\text{conv}(P)$ and that the convex hull is defined by the linear constraints

$$\begin{aligned}-x + y &\leq 1 \\ x + 2y &\leq 11 \\ 3x + 2y &\leq 21 \\ (6.5) \quad x + 3y &\geq 7.\end{aligned}$$

If we add the solution $(4, 3)$ to P , the convex hull of P is still given by (6.5) ([Figure 6.11](#)).

FIGURE 6.10 Convex hull of $\{(1, 2), (3, 4), (5, 3), (7, 0)\}$.

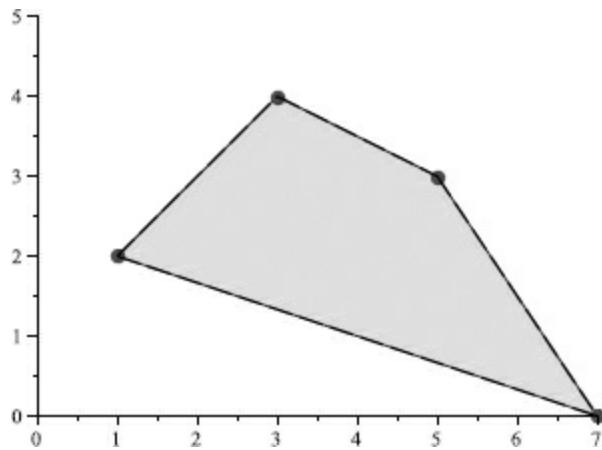
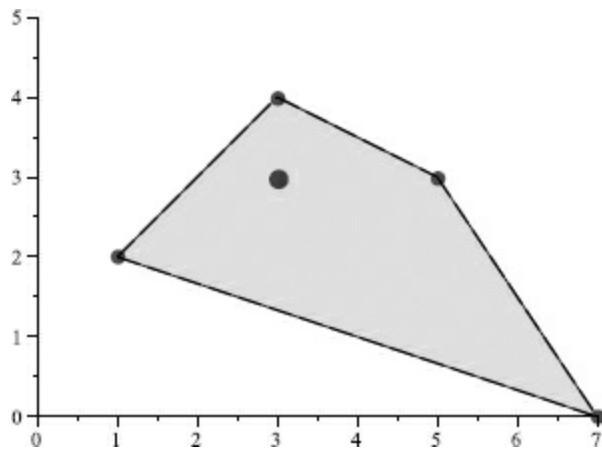


FIGURE 6.11 Convex hull of $\{(1, 2), (3, 4), (5, 3), (7, 0), (4, 3)\}$.



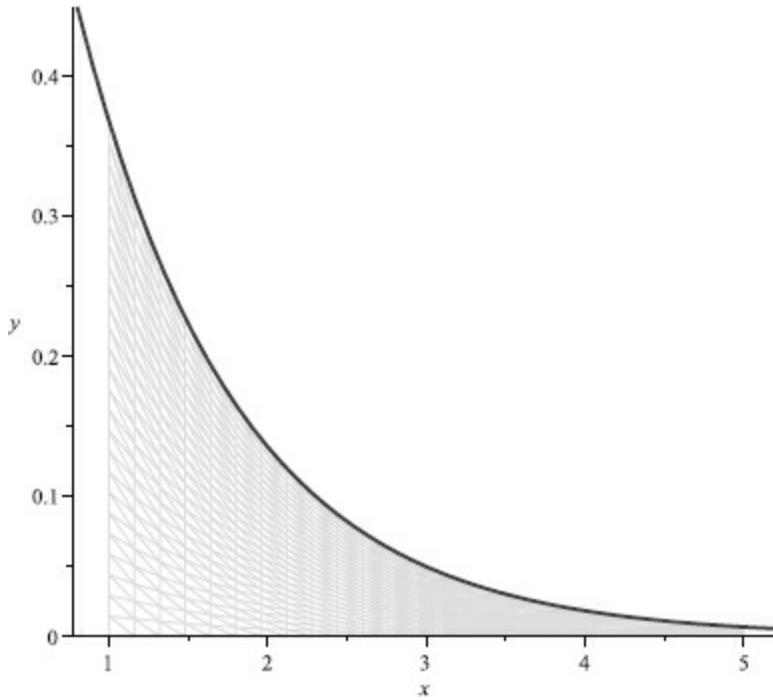
Theorem 6.1 Given a finite set of solutions S , the convex hull of S is the smallest convex set containing S ; that is, if $S \subseteq P$ and P is convex, then $\text{conv}(S) \subseteq P$. In fact, the convex hull of any finite set S can be written as a set of linear inequalities.

Directions of a Convex Set

We saw earlier that it is possible for an optimization problem to have an unbounded solution. Typically, for this to occur, the feasible region must be unbounded.

Unbounded Set A set S is **unbounded** if, for every $M > 0$, there exists an $\mathbf{x} \in S$ such that $\|\mathbf{x}\| > M$. Otherwise, S is **bounded**.

FIGURE 6.12 Set $\{x \geq 1, 0 < y < e^{-x}\}$ in Example 6.18.



■ EXAMPLE 6.18

The set $S = \{(x, y) : x \geq 1, 0 < y < e^{-x}\}$ is unbounded ([Figure 6.12](#)) since for every $M > 0$, the solution $\left(M + 1, \frac{1}{2}e^{-(M+1)}\right) \in S$ and

$$\left\| \left(M + 1, \frac{1}{2}e^{-(M+1)}\right) \right\| = \sqrt{(M + 1)^2 + \left(\frac{1}{2}e^{-(M+1)}\right)^2} > M + 1.$$

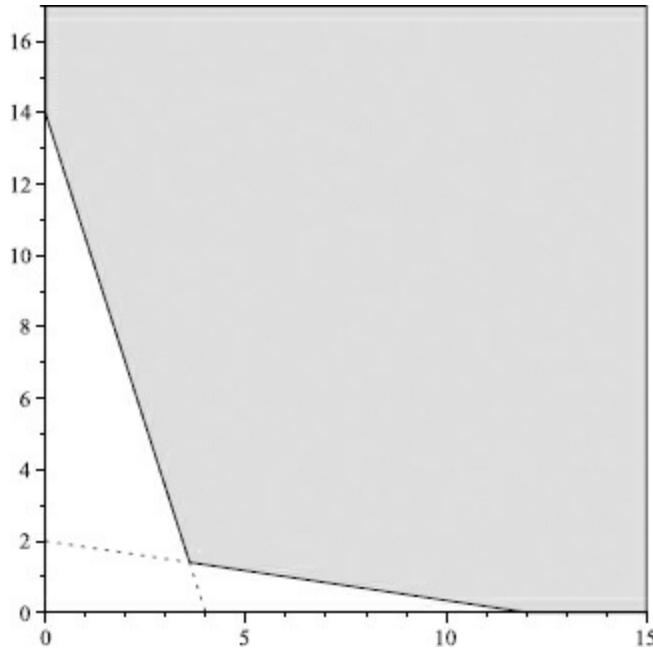
■ EXAMPLE 6.19

The set

$$S = \{(x, y) : 7x + 2y \geq 28, 2x + 12y \geq 24, x, y \geq 0\},$$

in [Figure 6.13](#), is unbounded because there exists a solution (x, y) where each x and y can be arbitrarily large. In this case, from any feasible solution (x, y) we can move in the direction $\mathbf{d} = (1, 1)$ to $(x + \lambda, y + \lambda)$, for $\lambda > 0$, and remain feasible.

FIGURE 6.13 Unbounded region in Example 6.19.



The previous examples illustrate the differences in unbounded sets between convex and nonconvex sets. In Example 6.19, we saw that there is a direction $\mathbf{d} = (1, 1)$ along which from any point in S we can move as far as we want. This observation gives rise to the following definition.

Unbounded Direction Given an unbounded set S , $\mathbf{d} \neq \mathbf{0}$ is an **unbounded direction** of S if, for all $\mathbf{x} \in S$, $\mathbf{x} + \lambda\mathbf{d} \in S$, for all $\lambda \geq 0$. Note that if \mathbf{d} is an unbounded direction of S , then so is $k\mathbf{d}$, for any $k > 0$.

■ EXAMPLE 6.20

For the set

$$S = \{(x, y) : 7x + 2y \geq 28, 2x + 12y \geq 24, x, y \geq 0\},$$

in Example 6.19, the directions $(1, 1)$ and $(2, 5)$ are both unbounded directions since, for example, if $(x, y) \in S$, then

$$7(x + 2\lambda) + 2(y + 5\lambda) = 7x + 2y + 14\lambda \geq 28, \text{ if } \lambda \geq 0 \quad \text{and}$$

$$2(x + 2\lambda) + 12(y + 5\lambda) = 2x + 12y + 64\lambda \geq 24, \text{ if } \lambda \geq 0.$$

Does every unbounded set S have an unbounded direction? No, the set $S = \{(x, y) : x \geq 1, 0 < y < e^{-x}\}$ given in Example 6.18 does not have an unbounded direction (see Exercise 6.13). However, if a set S is convex, the answer is yes. In fact, we've already run across this idea before. We saw earlier that maximizing a linear program was unbounded if $\mathbf{c}^T \mathbf{d} > 0$ for some

unbounded direction \mathbf{d} of the feasible region.

Lemma 6.2 A convex set S is unbounded if and only if S has at least one unbounded direction \mathbf{d} .

Convex and Concave Functions

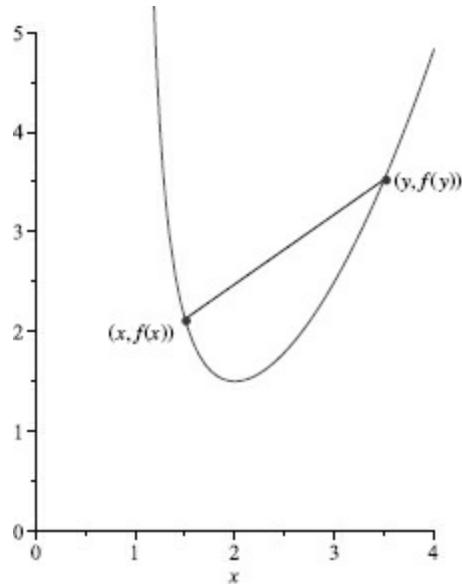
Let's now consider the objective function. What we'd ideally like is to optimize a function where if a solution \mathbf{y} is better than \mathbf{x} , and we start at \mathbf{x} , then the direction $\mathbf{d} = \mathbf{y} - \mathbf{x}$ is improving. This is akin to our story of the blindfolded person where there is only one peak on the island. There are such classes of functions, one of which is *convex function*.

Convex Function Given a convex set S , a function $f(\mathbf{x})$ is *convex* if for all solutions $\mathbf{x}, \mathbf{y} \in S$ and for all $\lambda \in [0, 1]$,

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}).$$

Geometrically, this means that the line segment connecting the solutions $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ lies on or above the graph of the function, as illustrated in [Figure 6.14](#).

[**FIGURE 6.14**](#) Example of a convex function.



We have seen such functions before in calculus. We learned that a twice-differentiable function was concave up if $f''(x) > 0$; for such functions, this is

exactly what a convex function is. Thus, $f(x) = x^2$ is a convex function since $f'(x) = 2 > 0$. Below is another example.

■ EXAMPLE 6.21

Let $f(x, y) = x^2 + y^2$. If $\mathbf{w} = (w_x, w_y)$ and $\mathbf{z} = (z_x, z_y)$, then

$$\begin{aligned} f(\lambda\mathbf{w} + (1 - \lambda)\mathbf{z}) &= f(\lambda w_x + (1 - \lambda)z_x, \lambda w_y + (1 - \lambda)z_y) \\ &= (\lambda w_x + (1 - \lambda)z_x)^2 + (\lambda w_y + (1 - \lambda)z_y)^2 \\ &= \lambda^2 w_x^2 + \lambda(1 - \lambda)w_x z_x + (1 - \lambda)^2 z_x^2 \\ &\quad + \lambda^2 w_y^2 + \lambda(1 - \lambda)w_y z_y + (1 - \lambda)^2 z_y^2. \end{aligned}$$

Suppose that $w_x \geq z_x$, $w_x \geq 0$, or $0 > z_x \geq w_x$. Then $w_x^2 \geq w_x z_x$ and so

$$\begin{aligned} (1 - \lambda)^2 z_x^2 + \lambda^2 w_x^2 + \lambda(1 - \lambda)w_x z_x &\leq \lambda^2 w_x^2 + \lambda(1 - \lambda)w_x^2 + (1 - \lambda)^2 z_x^2 \\ &= \lambda w_x^2 + (1 - \lambda)^2 z_x^2. \end{aligned}$$

If $z_x \geq w_x$, $z_x \geq 0$, or $0 > w_x \geq z_x$, then $z_x^2 \geq w_x z_x$ and so

$$\begin{aligned} (1 - \lambda)^2 z_x^2 + \lambda^2 w_x^2 + \lambda(1 - \lambda)w_x z_x &\leq (1 - \lambda)^2 z_x^2 + \lambda(1 - \lambda)z_x^2 + \lambda^2 w_x^2 \\ &= (1 - \lambda)z_x^2 + \lambda^2 w_x^2. \end{aligned}$$

We deduce similar inequalities involving w_y or z_y . Finally, since $\lambda \in [0, 1]$, we know that $\lambda^2 \leq \lambda$ and $(1 - \lambda)^2 \leq (1 - \lambda)$. Putting this together shows

$$\begin{aligned} f(\lambda\mathbf{w} + (1 - \lambda)\mathbf{z}) &= \lambda^2 w_x^2 + \lambda(1 - \lambda)w_x z_x + (1 - \lambda)^2 z_x^2 \\ &\quad + \lambda^2 w_y^2 + \lambda(1 - \lambda)w_y z_y + (1 - \lambda)^2 z_y^2 \\ &\leq \lambda(w_x^2 + w_y^2) + (1 - \lambda)(z_x^2 + z_y^2) \\ &= \lambda f(\mathbf{w}) + (1 - \lambda) f(\mathbf{z}), \end{aligned}$$

and so $f(x, y)$ is convex.

For multidimensional functions, there are extensions of the ideas in calculus that show a function is convex, but we do not address those here.

Since we hope to solve linear programs, determining whether a linear function is convex is useful.

Lemma 6.3 *All linear functions on a convex set S are convex.*

Proof Suppose $f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$ and $\mathbf{x}, \mathbf{y} \in S$. Then, given $\lambda \in [0, 1]$,

$$\begin{aligned} f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) &= \mathbf{c}^T(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) \\ &= \lambda\mathbf{c}^T\mathbf{x} + (1 - \lambda)\mathbf{c}^T\mathbf{y} \\ &= \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}). \end{aligned}$$

Convexity helps us immediately identify an improving direction.

Theorem 6.2 Consider minimizing a convex function $f(\mathbf{x})$ over a convex set S and suppose that $f(\mathbf{x}) < f(\mathbf{y})$. Then $\mathbf{d} = \mathbf{x} - \mathbf{y}$ is an improving direction from \mathbf{y} .

Proof Since f is a convex function,

$$\begin{aligned}f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) &= f(\mathbf{y} + \lambda(\mathbf{x} - \mathbf{y})) \\&\leq \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}) \\&= f(\mathbf{y}) + \lambda(f(\mathbf{x}) - f(\mathbf{y})) \\&< f(\mathbf{y}).\end{aligned}$$

But what about maximization problems? For these, we need a similar idea.

Concave Function Given a convex set S , a function $f(\mathbf{x})$ is *concave* if for all solutions $\mathbf{x}, \mathbf{y} \in S$ and for $\lambda \in [0, 1]$,

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \geq \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}).$$

Geometrically, this means the line segment connecting the solutions $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ lies on or below the graph of the function ([Figure 6.15](#)).

Similar to convex functions, concave functions have some nice properties.

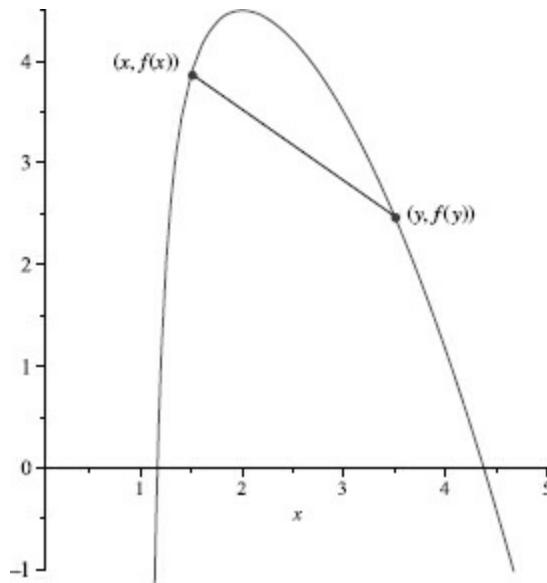
Lemma 6.4 All linear functions are concave.

Theorem 6.3 Consider maximizing a concave function $f(\mathbf{x})$ over a convex set S and suppose that $f(\mathbf{x}) > f(\mathbf{y})$, where $\mathbf{x}, \mathbf{y} \in S$. Then $\mathbf{d} = \mathbf{x} - \mathbf{y}$ is an improving direction from \mathbf{y} .

Convexity and Optimization

Given our earlier discussions, the following results stating why convexity is such an important idea should not be a surprise.

[**FIGURE 6.15**](#) Example of a concave function.



Theorem 6.4 Let S be a convex set.

1. Suppose we are minimizing a convex function $f(\mathbf{x})$ over S . If an improving search algorithm stops at a local minimum \mathbf{x} , then \mathbf{x} is a global minimum.
2. Suppose we are maximizing a concave function $f(\mathbf{x})$ over S . If an improving search algorithm stops at a local maximum \mathbf{x} , then \mathbf{x} is a global maximum.

Proof We prove the first statement and leave the second as an exercise. Suppose that an improving search algorithm has stopped at a local minimum solution \mathbf{x}^* and suppose that it is not the global minimum. There then exists a feasible solution $\mathbf{y} \in S$ such that $f(\mathbf{y}) < f(\mathbf{x}^*)$. Let $0 < \lambda \leq 1$ and consider the solution $\mathbf{x}^* + \lambda(\mathbf{y} - \mathbf{x}^*)$. Then

$$\begin{aligned} f(\mathbf{x}^* + \lambda(\mathbf{y} - \mathbf{x}^*)) &= f(\lambda\mathbf{y} + (1 - \lambda)\mathbf{x}^*) \leq \lambda f(\mathbf{y}) + (1 - \lambda)f(\mathbf{x}^*) \\ &< \lambda f(\mathbf{x}^*) + (1 - \lambda)f(\mathbf{x}^*) \\ &= f(\mathbf{x}^*). \end{aligned}$$

This implies that, for any neighborhood around \mathbf{x}^* , there exists a solution whose function value is smaller than $f(\mathbf{x}^*)$ (just take λ arbitrarily close to 1). In addition, since S is convex, this solution is in S , which contradicts \mathbf{x}^* being a local minimum.

Convexity and Linear Programs

We are especially interested in whether linear programs are convex. To begin our discussion, we introduce some definitions.

Hyperplane A *hyperplane* in \mathbb{R}^n is a set of the form $\{\mathbf{x} : \mathbf{p}^T \mathbf{x} = k\}$, where \mathbf{p} is a nonzero vector in \mathbb{R}^n and k is a scalar.

Half-space A *half-space* in \mathbb{R}^n is a set of points of the form $\{\mathbf{x} : \mathbf{p}^T \mathbf{x} \leq k\}$. Note that a hyperplane divides \mathbb{R}^n into two half-spaces.

Polyhedral Set A *polyhedral set*, or a **polyhedron**, is the intersection of a finite collection of half-spaces and hyperplanes. The hyperplanes $\mathbf{p}^T \mathbf{x} = k$ that define the half-spaces and hyperplanes are called the **defining hyperplanes** of the polyhedron.

Polytope A *polytope* is a bounded polyhedron.

■ EXAMPLE 6.22

Consider the linear program

$$\begin{aligned} \max \quad & 3x + 8y \\ \text{s.t.} \quad & x + 4y \leq 20 \\ & x + y \leq 9 \\ & 2x + 3y \leq 20 \\ & x, y \geq 0. \end{aligned}$$

The set

$$S = \left\{ \mathbf{x} = (x, y) : \begin{array}{l} x + 4y \leq 20 \\ x + y \leq 9 \\ 2x + 3y \leq 20 \\ x \geq 0 \\ y \geq 0 \end{array} \right\}$$

is a polyhedral set with five defining hyperplanes.

The next two lemmas will help us show that a polyhedral set is convex. Note that we've already hinted at the first lemma in Example 6.15.

Lemma 6.5 Let $S = \{\mathbf{x} : \mathbf{a}^T \mathbf{x} \leq b\}$ be a half-space in \mathbb{R}^n . Then S is convex.

Proof Let $\mathbf{x}, \mathbf{y} \in S$. We need to show that $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in S$, which follows from

$$\begin{aligned}
\mathbf{a}^T [\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}] &= \lambda \mathbf{a}^T \mathbf{x} + (1 - \lambda) \mathbf{a}^T \mathbf{y} \\
&\leq \lambda b + (1 - \lambda)b \\
&= b.
\end{aligned}$$

In fact, the set of all solutions that satisfy any linear constraint (equality, greater than, or less than) is convex (see Exercises 6.15 and 6.16). We also have the following fact, which we prove in Exercise 6.17.

Lemma 6.6 *If S_1, S_2 are convex sets, then $S = S_1 \cap S_2$ is also convex.*

Using these two lemmas, we have the following convexity result.

Theorem 6.5 *Let S be a polyhedral set. Then S is convex.*

We've already seen that linear functions are both convex and concave. These facts allow us to solve linear programs to global optimality through an improving search algorithm.

Linear programs can be solved to (global) optimality using an improving search algorithm, assuming the algorithm terminates.

6.4 FARKAS' LEMMA: WHEN CAN NO IMPROVING FEASIBLE DIRECTION BE FOUND?

For some problems, such as linear programs, criteria exist that help determine whether an improving feasible direction can be found. Consider the linear program

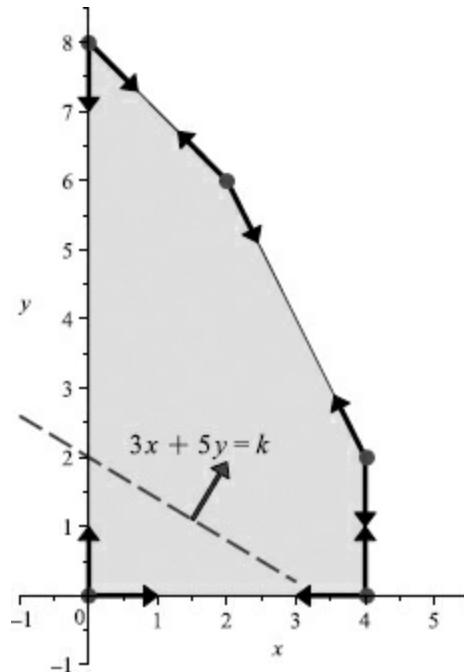
$$\begin{aligned}
\max \quad & 3x + 5y \\
\text{s.t.} \quad & \\
& x + y \leq 8 \\
& 2x + y \leq 10 \\
& x \leq 4 \\
& x, y \geq 0.
\end{aligned}$$

Its feasible region is shown in [Figure 6.16](#), along with an objective function contour (with its gradient vector) and some feasible directions at each of the corner points of the feasible region. Let's consider the five corner point

solutions $(0, 0)$, $(4, 0)$, $(4, 2)$, $(2, 6)$, and $(0, 8)$, where exactly two constraints or bounds are active. For each of these solutions, we can determine the system of inequalities any improving feasible direction would have to satisfy. To simplify our presentation, we shall rewrite the bounds $x \geq 0$, $y \geq 0$ as $-x \leq 0$, $-y \leq 0$ to have all constraints and bounds as less than or equal to constraints. For each solution, an improving direction $\mathbf{d} = (d_x, d_y)$ satisfies

$$(6.6) \quad \mathbf{c}^T \mathbf{d} = [3 \quad 5] \begin{bmatrix} d_x \\ d_y \end{bmatrix} = 3d_x + 5d_y > 0$$

FIGURE 6.16 Feasible region with feasible direction vectors and objective contour.



- At the solution $(0, 0)$, the active constraints are $-x \leq 0$, $-y \leq 0$; thus, all feasible directions $\mathbf{d} = (d_x, d_y)$ can be described by solutions to the system

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Note that any direction vector with nonnegative components is feasible, and, by (6.6), also improving.

- At the solution $(4, 0)$, the active constraints are $x \leq 4$ and $-y \leq 0$, giving the system

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

that describes all feasible directions at this solution. Note that the direction $\mathbf{d} = (-1, 0)$ is feasible but not improving, while the direction $\mathbf{d} = (0, 1)$ is both improving and feasible.

3. At the solution $(4, 2)$, the active constraints are $x \leq 4$ and $2x + y \leq 10$, giving the system

$$\begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

that describes all feasible directions at this solution. The direction $\mathbf{d} = (0, -1)$ is feasible but not improving, while $\mathbf{d} = (-1, 2)$ is both feasible and improving.

4. At the solution $(2, 6)$, the active constraints are $x + y \leq 8$ and $2x + y \leq 10$, giving the system

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

that describes all feasible directions at this solution. The direction $\mathbf{d} = (1, -2)$ is feasible but not improving, while $\mathbf{d} = (-1, 1)$ is both feasible and improving.

5. At the solution $(0, 8)$, the active constraints are $x + y \leq 8$ and $-x \leq 0$, giving the system

$$\begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

that describes all feasible directions at this solution. There are no improving feasible directions at this solution.

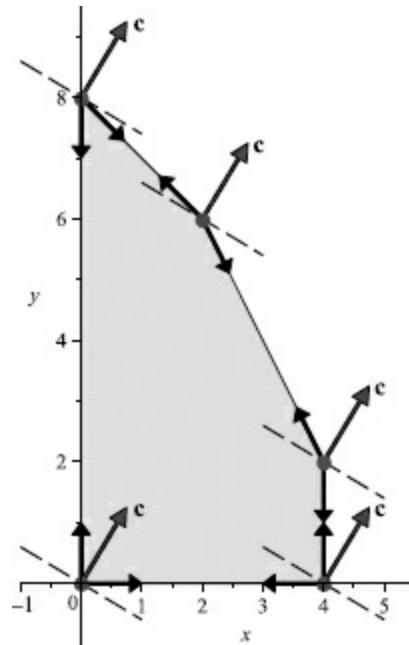
We can see graphically in [Figure 6.17](#) each of these characterizations of an improving feasible direction. At each corner point, the feasible directions along the constraints are provided, and all feasible directions point either “inside” the feasible region or along the boundary between these outer directions. Algebraically, this means that every feasible direction lies within the nonnegative span of the directions along the constraint. For example, at the corner point $(2, 6)$, every feasible direction can be written in the form

FIGURE 6.17 Gradient vector and feasible directions.

where $\alpha, \beta \geq 0$. This can be verified by noting that

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \left(\alpha \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ -2 \end{bmatrix} \right) = \begin{bmatrix} -\beta \\ -\alpha \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

In addition, at each corner point in [Figure 6.17](#) is the gradient vector \mathbf{c} and a dashed line that is perpendicular to \mathbf{c} . Any improving direction must be on the “gradient side” of the dashed line since these direction vectors make an acute angle with \mathbf{c} . Thus, there exists an improving feasible direction at a corner point if and only if there are directions pointing inside the feasible region (or along its boundary) and on the gradient side of the line. We can easily see that $(0, 8)$ is the only such solution that has no improving feasible direction.



$$\mathbf{d} = \alpha \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

Each of these systems describing the improving feasible directions at the given solution is written algebraically as

$$(6.7) \quad A\mathbf{d} \leq 0, \quad \mathbf{c}^T \mathbf{d} > 0,$$

where A is the matrix of the constraint coefficients corresponding to active constraints at a given solution. In 1894, G. Farkas examined the question of when such a system (6.7) has a solution. To understand his answer, let's consider what the matrix A^T represents geometrically. For example, at the

solution $(2, 6)$, the system(6.7) is

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad 3d_x + 5d_y > 0.$$

The matrix A^T is

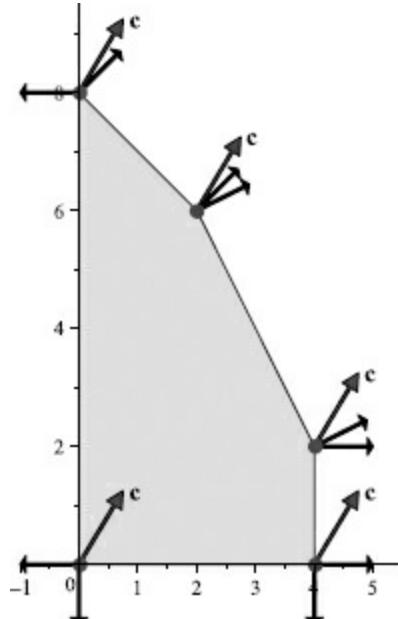
$$A^T = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}.$$

Note that now the columns of A^T are the constraint coefficients of each active constraint; in other words, the columns of A^T are the normal vectors of each active constraint, and hence perpendicular to the constraint lines. This tells us that $A^T y$ represents a linear combination of the normal vectors. Now, consider [Figure 6.18](#) that contains at each corner point the gradient and the normal vectors of each active constraint. In each corner point except for $(0, 8)$, the gradient vector is not contained within the positive span of the normal vectors, while it is at $(0, 8)$. Algebraically, this means that at $(0, 8)$ the system

$$A^T y = c, \quad y \geq 0$$

has a solution, but at the other corner points it does not. Furthermore, recall that $(0, 8)$ is the lone corner point from which an improving feasible direction cannot be found. Such an observation led Farkas to the following result.

FIGURE 6.18 Gradient vector and constraint normal vectors.



Lemma 6.1 (Farkas' Lemma) Given an $m \times n$ matrix A and n -

dimensional vector \mathbf{c} , one and only one of the following two systems has a solution.

$$\text{System 1 : } \mathbf{A}\mathbf{d} \leq \mathbf{0} \text{ and } \mathbf{c}^T \mathbf{d} > 0.$$

$$\text{System 2 : } \mathbf{A}^T \mathbf{y} = \mathbf{c} \text{ and } \mathbf{y} \geq \mathbf{0}.$$

We leave the proof for later. As noted in the above example, geometrically Farkas' lemma characterizes when a feasible solution to a linear program has an improving feasible direction. There are many versions of this result, such as when the linear system in System 1 is an equality or when the right-hand side in System 1 is a general vector \mathbf{b} . In fact, given that all local optimal solutions to a linear program are global solutions, Farkas' lemma provides an **Optimality Condition** for linear programs, just like those found in Chapter 5 for the minimum spanning tree problem. We revisit this idea in Chapter 9.

Summary

In this chapter, we've seen a basic approach to finding an algorithm that solves mathematical programs. In particular, we've explored aspects of this approach that are specific to linear programs. In the end, we've found that this general approach can be successful in solving linear programs.

But we're far from done. As we've seen, actually finding a feasible improving direction is not necessarily easy. We've also ignored the fact that we actually have to start at a feasible solution and guarantee that the algorithm will terminate. We will be discussing these topics in the next few chapters, as we come up with a very specific version of the general improving search algorithm to solve linear programs.

EXERCISES

6.1 For each of the following objective functions and current solutions, determine whether the given directions are improving or not.

- (i) maximize $3x_1 + 4x_2 - 6x_3$ at the point $\mathbf{x} = (2, 3, 1)$.
 - (ii) minimize $5x_1 - 2x_2 + 3x_3$ at the point $\mathbf{x} = (-1, 2, 1)$.
- (a)** $\mathbf{d} = (1, 2, 3)$.

(b) $\mathbf{d} = (1, 0, -2)$.

(c) $\mathbf{d} = (0, -1, -1)$.

6.2 Each of the following shows the sequence of directions and steps employed by an improving search algorithm that began at the point $\mathbf{x}^{(0)} = (1, 0, -2)$. Compute the sequence of points visited by the search.

(a) $\mathbf{d}^{(1)} = (3, 2, 1), \lambda_1 = 2; \mathbf{d}^{(2)} = (1, 0, -3), \lambda_2 = 4; \mathbf{d}^{(3)} = (-1, 4, 0), \lambda_3 = \frac{1}{2}$.

(b) $\mathbf{d}^{(1)} = (4, -1, 2), \lambda_1 = 3; \mathbf{d}^{(2)} = (0, 2, -1), \lambda_2 = 5; \mathbf{d}^{(3)} = (1, 2, 1), \lambda_3 = 1$.

6.3 For each of the following objective functions and current solutions \mathbf{x} , use the gradient (or negative gradient for minimization problems) as our search direction to determine the (positive) step size λ_{\max} that will yield the largest improvement in our objective value or show that $\lambda_{\max} \rightarrow \infty$.

(a) $\min 3x^2 - 2xy + y^2 - 10; \mathbf{x} = (1, 3)$.

(b) $\min x^2 - 3xy - 2y^2; \mathbf{x} = (2, -1)$.

(c) $\max x^2 + 12xy - y^2; \mathbf{x} = (-1, 1)$.

(d) $\max x^2 - 8xy - 10xz - yz - y^2 - 3z^2; \mathbf{x} = (1, 1, 1)$.

6.4 Each of the following shows the sequence of solutions visited by an improving search algorithm. Assuming that the step size at each iteration is $\lambda = 1$, determine the search directions at each iteration.

(a) $\mathbf{x}^{(0)} = (1, 0), \mathbf{x}^{(1)} = (2, -1), \mathbf{x}^{(2)} = (0, 3)$.

(b) $\mathbf{w}^{(0)} = (-1, 2, -3), \mathbf{w}^{(1)} = (1, 1, 4), \mathbf{w}^{(2)} = (4, -3, 5)$.

(c) $\mathbf{w}^{(0)} = (10, 3, 1), \mathbf{w}^{(1)} = (12, 7, 4), \mathbf{w}^{(2)} = (4, 3, 8), \mathbf{w}^{(3)} = (6, 2, 7)$.

6.5 Consider a mathematical program with constraints

$$2x_1 + 4x_2 - x_3 \leq 20$$

$$x_1, x_2, x_3 \geq 0.$$

Determine the maximum step size (possibly $+\infty$) that preserves feasibility in the direction indicated from the solution specified. Also, indicate whether that step indicates that the model is unbounded, assuming that directions improve everywhere.

(a) $\mathbf{d} = (1, 2, 1)$ from $\mathbf{x} = (2, 2, 1)$.

(b) $\mathbf{d} = (2, -1, -3)$ from $\mathbf{x} = (3, 1, 5)$.

(c) $\mathbf{d} = (1, -1, 3)$ from $\mathbf{x} = (10, 0, 6)$.

6.6 Consider a mathematical program with constraints

$$x_1 + 3x_2 + 2x_3 \leq 15$$

$$2x_1 - x_2 + x_3 \geq 5$$

$$x_1, x_3 \geq 0.$$

Determine the maximum step size (possibly $+\infty$) that preserves feasibility in the direction indicated from the solution specified. Also, indicate whether that step indicates that the model is unbounded, assuming that directions improve everywhere.

(a) $\mathbf{d} = (1, 2, 1)$ from $\mathbf{x} = (2, 2, 3)$.

(b) $\mathbf{d} = (4, 1, 2)$ from $\mathbf{x} = (10, 0, 2)$.

(c) $\mathbf{d} = (-2, 1, 1)$ from $\mathbf{x} = (2, 1, 5)$.

(d) $\mathbf{d} = (3, 4, 0)$ from $\mathbf{x} = (5, 1, 1)$.

6.7 Consider the problem

$$\max x_1$$

s.t.

$$x_1^2 + x_2^2 \leq 9$$

$$x_1^2 \geq 1.$$

Graph the feasible region and identify all local maxima for this problem. Which ones are global maxima?

6.8 Determine which of the constraints

[i] $3y_1 - 2y_2 \leq 9$

[ii] $2y_1^2 - y_1y_2 - y_2^2 \geq 5$

[iii] $y_1 + y_2 = 3$

[iv] $y_1 \geq 0$

[v] $y_2 \geq 0$

are active at each of the following solutions.

(a) $\mathbf{y} = (3, 0)$.

(b) $\mathbf{y} = (2, 1)$.

6.9 Determine whether each of the directions specified is feasible at the solution indicated if the feasible region is defined by the constraints

$$6x_1 + 2x_2 - x_3 = 15$$

$$2x_1 + 3x_2 + 5x_3 \leq 21.$$

(a) $\mathbf{d} = (2, -1, 0)$ at the solution $(3, 0, 3)$.

(b) $\mathbf{d} = (-2, 1, 0)$ at the solution $(3, 0, 3)$.

(c) $\mathbf{d} = (1, -2, 2)$ at the solution $(2, 2, 1)$.

6.10 State all conditions that must be satisfied by a feasible direction \mathbf{d} at the solution indicated to each of the following systems of linear constraints.

(a) Constraints

$$\begin{aligned} 2x_1 + x_3 &= 12 \\ x_1 + 4x_2 - x_3 &\leq 10 \\ x_1, x_2 &\geq 0 \end{aligned}$$

at the solution $\mathbf{x} = (5, 0, 2)$.

(b) Same constraints as (a) but at the solution $\mathbf{x} = (6, 1, 0)$.

(c) Constraints

$$\begin{aligned} x_1 + x_2 &\leq 8 \\ 3x_1 - x_2 &\geq 12 \\ x_1 - 2x_2 &\geq 2 \end{aligned}$$

at the solution $\mathbf{x} = (2, -6)$.

(d) Same constraints as (c) at the solution $\mathbf{x} = (6, 2)$.

6.11 Consider the following linear program

$$\begin{aligned} \max \quad & 6x_1 + 5x_2 \\ \text{s.t.} \quad & \\ & 5x_1 + 2x_2 \leq 34 \\ & x_1 - x_2 \geq -3 \\ & x_1, x_2 \geq 0. \end{aligned}$$

(a) Show that the directions $\mathbf{d}^{(1)} = (3, 1)$ and $\mathbf{d}^{(2)} = (-2, 5)$ are both improving directions at every feasible solution.

(b) Beginning at $\mathbf{x}^{(0)} = (0, 0)$, execute Algorithm 6.2 using only these two directions. Continue until neither direction is both improving and feasible.

(c) Show in a two-dimensional plot the feasible region of this problem. Then plot the path of your search in part (b).

6.12 Consider the problem of minimizing $f(\mathbf{x}) = x^2$, starting from an initial

solution $x^0 = 2$. Clearly, since $x = 0$ is the global minimum (and only local minimum), an improving direction is $d = -1$. Suppose that during iteration k , we choose our step size $\lambda = 2^{-(k+1)}$. Show that the sequence of solutions x^0, x^1, \dots , and so on converges, but does not converge to the optimal solution.

6.13 Verify that the unbounded set

$$S = \{(x, y) : x \geq 1, 0 < y < e^{-x}\}$$

does not have an unbounded direction. (*Hint:* Any potential unbounded direction $\mathbf{d} = (d_x, d_y)$ must have $d_x \geq 0$ and $d_y \geq 0$).

6.14 Determine whether the following regions are convex or not. If not, indicate two solutions \mathbf{x} and \mathbf{y} that violate the definition.

(a) Region

$$\begin{aligned} x_1^2 + x_2^2 &\geq 9 \\ x_1 + x_2 &\leq 10 \\ x_1, x_2 &\geq 0. \end{aligned}$$

(b) Region

$$\begin{aligned} x_1^2 + x_2^2 &\leq 9 \\ x_1 + x_2 &\leq 10 \\ x_1, x_2 &\geq 0. \end{aligned}$$

(c) Region

$$\begin{aligned} x_1 + x_2 + 3x_3 &\leq 10 \\ 2x_1 + 3x_2 + x_3 &\geq 5 \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

(d) Region

$$\begin{aligned} x_1 + 3x_2 + 2x_3 + x_4 &= 10 \\ x_2 + x_3 + 2x_4 &\leq 8 \\ 3x_1 + x_2 + 2x_3 &\geq 6 \\ x_1, x_2, x_3, x_4 &\geq 0, \text{ integer.} \end{aligned}$$

6.15 Show that the set $S = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T \mathbf{x} = b\}$ is convex.

6.16 Show that the set $S = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}^T \mathbf{x} \geq b\}$ is convex.

6.17 Let S_1, S_2 be convex sets. Show that $S = S_1 \cap S_2$ is also convex.

6.18 Consider the functions $f_1(x) = 10 - x$ and $f_2 = f_2 = \frac{1}{3}x + 2$, and let

$$g(\mathbf{x}) = \max\{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\}.$$

(a) Graph $g(x)$.

(b) Prove that $g(x)$ is convex.

6.19 Let f_1, \dots, f_m be linear functions in \mathbb{R}^n and suppose

$$g(\mathbf{x}) = \max\{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\}.$$

Prove that $g(\mathbf{x})$ is convex.

6.20 Let f_1, \dots, f_m be convex functions in \mathbb{R}^n and suppose

$$g(\mathbf{x}) = \max\{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\}.$$

Prove that $g(\mathbf{x})$ is convex.

6.21 Prove that the affine function $f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} + k$ is both convex and concave.

6.22 Prove that a function $f(\mathbf{x})$ is both convex and concave on \mathbb{R}^n if and only if $f(\mathbf{x}) = \mathbf{a}^T \mathbf{x} + k$ for some constant vector \mathbf{a} and scalar k .

6.23 Prove that a function $f(\mathbf{x})$ is concave if and only if $-f(\mathbf{x})$ is convex.

6.24 Suppose we are maximizing a concave function $f(\mathbf{x})$ over a convex set S . Prove that if an improving search algorithm stops at a local maximum \mathbf{x} , then \mathbf{x} is a global maximum.

6.25 Let $f(\mathbf{x})$ be a convex function.

(a) Show that if $k > 0$, then $kf(\mathbf{x})$ is convex.

(b) Show that if $k < 0$, then $kf(\mathbf{x})$ is concave.

6.26 Let $f(\mathbf{x})$ is a convex function in \mathbb{R}^n Prove that the set

$$S = \{\mathbf{x} : f(\mathbf{x}) \leq k\}$$

is a convex set for all real numbers k .

6.27 Let

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 2 \\ -1 & 4 \end{bmatrix}.$$

For each of the following vectors, determine which of the systems from Farkas' lemma has a solution and illustrate this geometrically.

(a) $\mathbf{c} = (1, 4)$.

(b) $\mathbf{c} = (-1, 1)$.

(c) $\mathbf{c} = (2, 1)$

6.28 Suppose that the following system has no solution

$$A\mathbf{x} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0}, \quad \mathbf{c}^T \mathbf{x} > 0.$$

Use Farkas' lemma to derive another system that must have a solution.

6.29 Suppose that the following system has no solution

$$A\mathbf{x} \geq \mathbf{0}, \quad \mathbf{c}^T \mathbf{x} < 0.$$

Use Farkas' lemma to derive another system that must have a solution.

6.30 Given a finite set of points S , show that if $S \subseteq P$ and P is convex, then $\text{conv}(S) \subseteq P$.

CHAPTER 7

GEOMETRY AND ALGEBRA OF LINEAR PROGRAMS

Now that we've explored the construction of an optimization algorithm in Chapters 5 and 6, we turn our focus to how linear programs are solved. We could just use an improving search algorithm to solve linear programs, but this could cause problems since the general algorithm does not specify how to choose an improving feasible direction. How we can do this is the basis of this chapter.

One way to refine our previous approaches for linear programs is to characterize the structure of potential optimal solutions. For example, in Chapters 1 and 6, we saw that optimal solutions to two-variable linear programs occurred at "corner points," but is this true if we have more than two variables, and if so, how do we identify them so that we can generate directions that head toward them? This is a problem of both geometry and algebra. In fact, what we want is to both efficiently (in terms of time and space) and easily describe such solutions so that we can refine our improving search approach to look for these characteristics.

In addition, because linear programs can have different types of constraints and bounds, do we need to have separate algorithms for each, or is there a common form that suffices? If this is the case, what form should we work with? These are questions we should consider as we refine our algorithmic approach to linear programs.

In this chapter, we consider what is a "corner point" of a polyhedral set for more than two dimensions, both algebraically and geometrically. We then address some other aspects of polyhedral sets, such as degeneracy and adjacency, that will have implications when we solve linear programs. The fundamental theorem of linear programming follows and tells us what an optimal solution (if it exists) can look like. Finally, we consider a "canonical

form” of linear programs and characterize corner points for this case.

7.1 GEOMETRY AND ALGEBRA OF “CORNER POINTS”

In various examples from Chapters 1 and 6, we found optimal solutions to linear programs at “corner points”, or where two constraints intersect. A natural question is “Does every linear program in n variables that has a finite optimal solution have, as one of its global optimum, a “corner point,” whatever that is in n dimensions?” If the answer is “yes,” the implications are huge; if we know how to describe “corner points,” then we have to use only search directions that point toward them; there would be no sense in looking at other solutions. Of course, before we can answer this question, we have to understand what a corner point is.

Basic Solutions of Polyhedral Sets

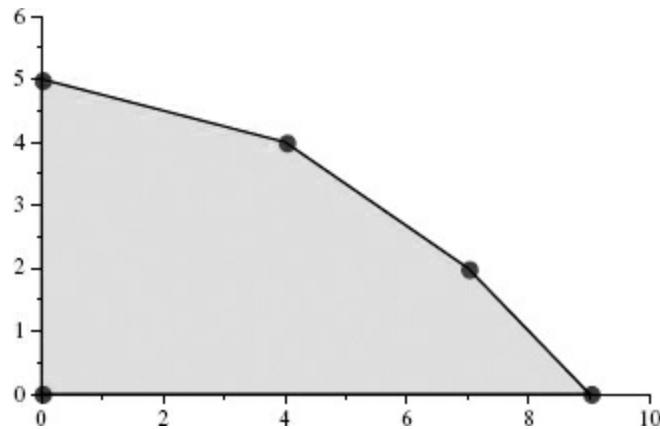
Let’s consider the region in \mathbb{R}^2 defined by the inequalities

$$(7.1) \quad S = \left\{ \mathbf{x} = (x, y) : \begin{array}{l} x + 4y \leq 20 \\ x + y \leq 9 \\ 2x + 3y \leq 20 \\ x \geq 0 \\ y \geq 0 \end{array} \right\}.$$

This polyhedral set is shown in [Figure 7.1](#). Note that the corner points are $(0, 0)$, $(9, 0)$, $(7, 2)$, $(4, 4)$, and $(0, 5)$ and that each occurs at the intersection of two lines, or equivalently, each corner point is active at two distinct constraints. Furthermore, the intersecting lines at each corner point are linearly independent, so that their intersection occurs at exactly one point. It is this algebraic notion that generalizes to higher dimensions. An important definition is the following.

Linearly Independent Hyperplanes Any collection of defining hyperplanes of a polyhedron is *linearly independent* if the coefficient matrix associated with the left-hand side of these equations has full row rank; that is, the rank of the matrix equals the number of rows.

FIGURE 7.1 Set S given in (7.1), with its corner points marked.



■ EXAMPLE 7.1

Consider the polyhedral set S given in (7.1). The set of defining hyperplanes

$$S_1 = \{2x + 3y = 20, x + y = 9\}$$

is linearly independent because the coefficient matrix

$$B = \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix}$$

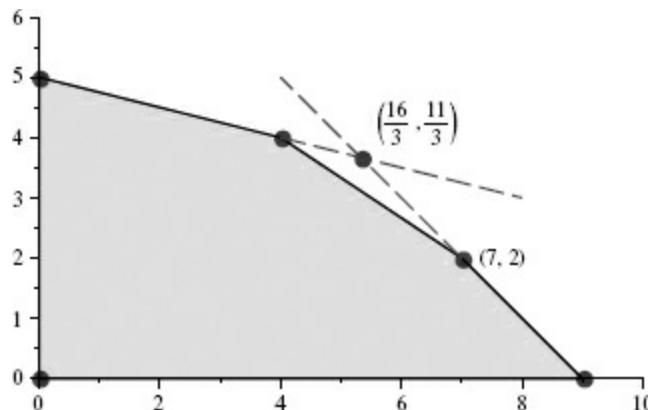
has full row rank (i.e., rank of 2). Because B is a square matrix with full row rank, there exists exactly one solution to the system $B\mathbf{x} = \mathbf{b}$ for any vector $\mathbf{b} \in \mathbb{R}^2$.

Given a polyhedral set $S \subset \mathbb{R}^n$, if we have a set of n linearly independent defining hyperplanes, there is a unique intersection point, and we can classify such solutions.

Basic Solution Let S be a polyhedral set defined by linear inequality and equality constraints. Solution \mathbf{x} is a *basic solution* if (a) \mathbf{x} satisfies all equality constraints of S and (b) at least n of the constraints of S are active at \mathbf{x} and are linearly independent.

Basic Feasible Solution If \mathbf{x} also satisfies all constraints of S , then \mathbf{x} is a *basic feasible solution*.

FIGURE 7.2 Feasible region to Example 7.2, with basic and basic feasible solutions.



■ EXAMPLE 7.2

Again consider the set S given in (7.1). The solution $(7, 2)$ is a basic feasible solution, while the solution $(\frac{16}{3}, \frac{11}{3})$ is a basic solution because the constraints $x + 4y \leq 20$ and $x + y \leq 9$ are active at this solution, but it does not satisfy $2x + 3y \leq 20$ ([Figure 7.2](#)).

A basic feasible solution matches our intuition regarding how to extend corner points to more variables. In fact, this definition formalizes our thinking of corner points as the intersection of constraints.

Of course, basic feasible solutions provide an algebraic definition, but what about a geometric interpretation? This notion of a basic feasible solution rests exclusively among convex sets containing half-spaces and hyperplanes; however, “special” solutions can be found in more general convex sets. Think back to the definition of convex sets given in Chapter 6: a set S is *convex* if, for all $\mathbf{x}, \mathbf{y} \in S$, the points $\lambda\mathbf{x} + (1 - \lambda)\mathbf{y} \in S$ for all $\lambda \in [0, 1]$. In other words, given two solutions in S , all solutions on the line segment connecting them are also in S . This definition leads us to classify points that have a special property.

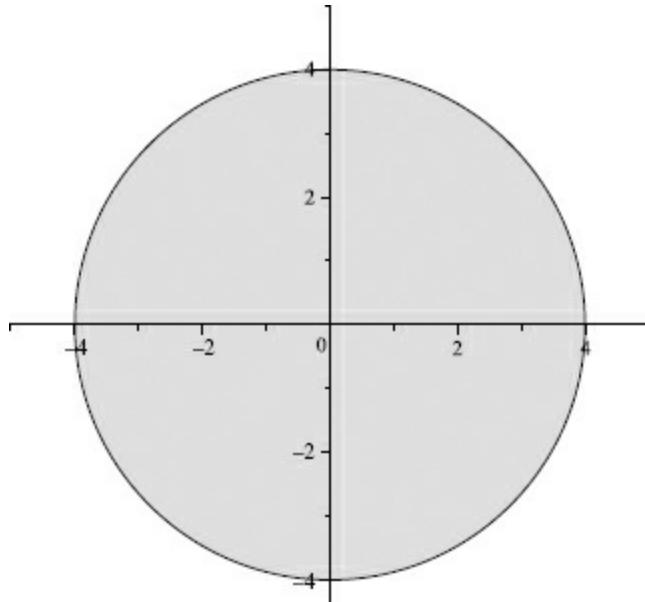
Extreme Point Given a convex set S , a solution $\mathbf{x} \in S$ is an *extreme point* if there does not exist two distinct solutions $\mathbf{y}, \mathbf{z} \in S$ such that \mathbf{x} is on the line segment joining \mathbf{y} and \mathbf{z} ; that is, there does not exist a $\lambda \in (0, 1)$ such that $\mathbf{x} = \lambda\mathbf{y} + (1 - \lambda)\mathbf{z}$.

■ EXAMPLE 7.3

Consider again the polyhedral set described in (7.1) and shown in [Figure 7.1](#).

Note that the corner points (basic feasible solutions) $(0, 0)$, $(9, 0)$, $(7, 2)$, $(4, 4)$,

FIGURE 7.3 $S = \{(x, y) : x^2 + y^2 \leq 16\}$.



and $(0, 5)$ are extreme points since none of these solutions are on a line segment connecting two other solutions in S . Are there any other extreme points? No.

This definition of extreme point is a general one, working for any convex set.

■ EXAMPLE 7.4

Consider the set $S = \{(x, y) : x^2 + y^2 \leq 16\}$ from Example 6.14, where we saw that S is convex. It is easy to see from [Figure 7.3](#) that the extreme points are all the solutions on the boundary $B = \{(x, y) : x^2 + y^2 = 16\}$, which are the solutions that are not on any line segment joining two other solutions in the set.

Not all convex sets have extreme points; for example, \mathbb{R}^n is convex (check this!), but there are no extreme points. In fact, not every polyhedral set has an extreme point—the half-space $S = \{(x, y) : x + y \geq 1\}$ does not have any. We later address the question of when a polyhedral set contains an extreme point.

It is important to note that the algebraic definition of a basic feasible

solution and the geometric one of an extreme point are in fact equivalent for polyhedral sets.

Theorem 7.1 Suppose S is a polyhedral set in \mathbb{R}^n . Then \mathbf{x} is an extreme point of S if and only if \mathbf{x} is a basic feasible solution.

Proof Suppose \mathbf{x} is a basic feasible solution. Then \mathbf{x} lies on n linearly independent defining hyperplanes of S . Let G be the $(n \times n)$ matrix of constraint coefficients of any n of these hyperplanes and \mathbf{g} the corresponding vector of right-hand-side constants so that $G\mathbf{x} = \mathbf{g}$. Note that G has full rank. Assume \mathbf{x} is not an extreme point. Then there exists solutions $\mathbf{x}_1 \neq \mathbf{x}_2$ both in S such that

$$\mathbf{x} = \lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2,$$

where $0 < \lambda < 1$. Since $G\mathbf{x}_1 = G\mathbf{x}_2 = \mathbf{g}$ (why?), we have

$$\begin{aligned}\mathbf{g} &= G\mathbf{x} \\ &= G(\lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2) \\ &= G(\mathbf{x}_2 + \lambda(\mathbf{x}_1 - \mathbf{x}_2)) \\ &= G\mathbf{x}_2 + \lambda G(\mathbf{x}_1 - \mathbf{x}_2) \\ &= \mathbf{g} + \lambda G(\mathbf{x}_1 - \mathbf{x}_2).\end{aligned}$$

Hence, $G(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{0}$. But, since G has rank n , the only solution to this equation is $\mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0}$, and hence $\mathbf{x}_1 = \mathbf{x}_2$, which is a contradiction.

Suppose that \mathbf{x} is not a basic feasible solution, so that the maximum number of linearly independent defining hyperplanes active at \mathbf{x} is $r < n$. As discussed before, let these active constraints be $G\mathbf{x} = \mathbf{g}$, where G is now a $r \times n$ matrix. Let $\mathbf{d} \neq \mathbf{0}$ be a solution to $G\mathbf{d} = \mathbf{0}$ (why does such a solution exist?) and consider the solutions $\mathbf{x}_1 = \mathbf{x} + \epsilon \mathbf{d}$ and $\mathbf{x}_2 = \mathbf{x} - \epsilon \mathbf{d}$, where $\epsilon > 0$. Note that if ϵ is small enough, both $\mathbf{x}_1, \mathbf{x}_2 \in S$. Thus, $\bar{\mathbf{x}} = \frac{1}{2}\mathbf{x}_1 + \frac{1}{2}\mathbf{x}_2$ and hence is not an extreme point of S .

We have two different, yet equivalent, ways to describe the same idea—one geometric (extreme point) and one algebraic (basic feasible solution). **We use these terms interchangeably throughout the rest of this book.**

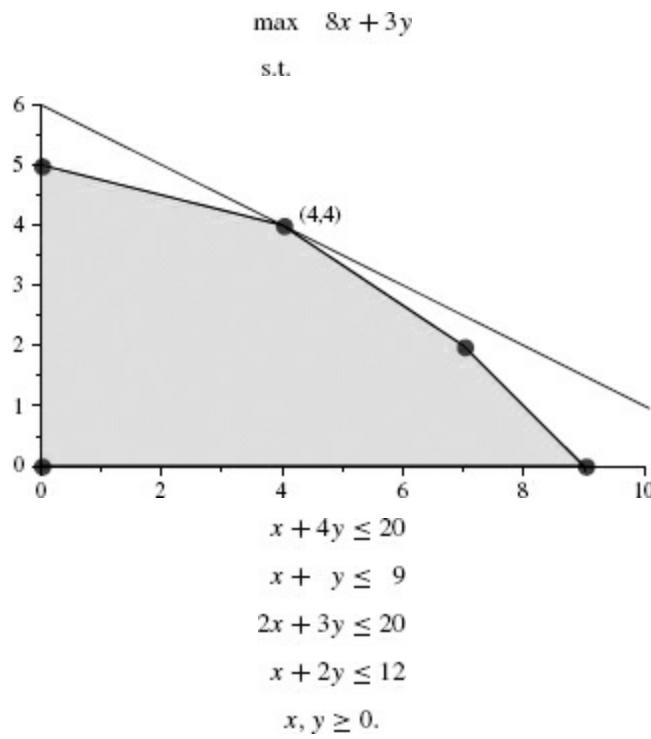
Degeneracy

The term “at least” appears in our definitions of basic (feasible) solutions, and it is important that we consider what happens if “too many” constraints are active.

■ EXAMPLE 7.5

Consider the linear program

FIGURE 7.4 Feasible region to Example 7.5.



Its feasible region is shown in [Figure 7.4](#). The active constraints at $(4, 4)$ are

$$\begin{aligned} x + 4y &\leq 20 \\ 2x + 3y &\leq 20 \\ x + 2y &\leq 12. \end{aligned}$$

Degenerate Solution A basic feasible solution \mathbf{x} of a polyhedral set S in \mathbb{R}^n is *degenerate* if there are more than n constraints active at \mathbf{x} . A basic feasible solution that is not degenerate is called **nondegenerate**.

The notion of degeneracy is important as we discuss algorithms for solving linear programs, and we will return to it later. However, note that, unlike the case in Example 7.5, it is not necessarily true that degeneracy can be eliminated by the removal of a redundant constraint; see Exercise 7.10 for a

simple example.

Adjacency

The set S given by (7.1) and shown in [Figure 7.1](#) demonstrates that each extreme point is “connected” to another extreme point via one defining hyperplane. This important idea can be generalized to multiple dimensions.

Edge of Polyhedron An *edge* of a polyhedral set S is the set of solutions formed by the intersection of $(n - 1)$ linearly independent defining hyperplanes.

In other words, an edge of a polyhedral set S is a line segment that satisfies $(n - 1)$ constraints at equality. Note that, when compared to extreme points, edges have one “degree of freedom,” since we are solving $n - 1$ equations for n variables; we have the freedom to fix the value of one variable, at which point the remaining $(n - 1)$ variables become fixed. In fact, they are line segments in \mathbb{R}^n . For extreme points, all variables are fixed once we know the set of hyperplanes that are active at that point.

■ EXAMPLE 7.6

In the polyhedral set S given in (7.1), each defining hyperplane defines an edge. Consider the edge consisting of all feasible solutions of S on the hyperplane $2x + 3y = 20$, which includes the extreme points $(4, 4)$ and $(7, 2)$. This line has slope $m = -\frac{2}{3}$ and, using the point $(4, 4)$, can be rewritten as

$$y - 4 = -\frac{2}{3}(x - 4).$$

In vector form, this line is

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ -\frac{2}{3} \end{bmatrix} t,$$

where t is the “degree of freedom” parameter. We can scale the direction to rewrite this line as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 3 \\ -2 \end{bmatrix} t.$$

Thus, when $t = 0$, we are at $(4, 4)$, and when $t = 1$, we are at $(7, 2)$. If we instead base our calculations on $(7, 2)$, we get

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ -\frac{2}{3} \end{bmatrix} t \\ = \begin{bmatrix} 7 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ 2 \end{bmatrix} t.$$

Thus, we have written this edge as the line segment joining $(4, 4)$ and $(7, 2)$, using two opposite directions.

In a similar manner, the remaining edges of S can be written as follows:

Solutions on Line Segment	Representations (for $0 \leq \lambda \leq 1$)
$(0, 0)$ and $(0, 5)$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 5 \end{bmatrix} \lambda, \begin{bmatrix} 0 \\ 5 \end{bmatrix} + \begin{bmatrix} 0 \\ -5 \end{bmatrix} \lambda$
$(0, 5)$ and $(4, 4)$	$\begin{bmatrix} 0 \\ 5 \end{bmatrix} + \begin{bmatrix} -4 \\ -1 \end{bmatrix} \lambda, \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} -4 \\ 1 \end{bmatrix} \lambda$
$(4, 4)$ and $(7, 2)$	$\begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 3 \\ -2 \end{bmatrix} \lambda, \begin{bmatrix} 7 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ 2 \end{bmatrix} \lambda$
$(7, 2)$ and $(9, 0)$	$\begin{bmatrix} 7 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} \lambda, \begin{bmatrix} 9 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 \\ 2 \end{bmatrix} \lambda$
$(9, 0)$ and $(0, 0)$	$\begin{bmatrix} 9 \\ 0 \end{bmatrix} + \begin{bmatrix} -9 \\ 0 \end{bmatrix} \lambda, \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 9 \\ 0 \end{bmatrix} \lambda$

Finally, using edges we can talk about this idea of “connected.”

Adjacent Extreme Points Two extreme points of a polyhedral set S in \mathbb{R}^n are **adjacent** if the line segment joining them is an edge of S . Equivalently, two extreme points of S are adjacent if there are $n - 1$ linearly independent defining hyperplanes of S that are active at both extreme points.

■ EXAMPLE 7.7

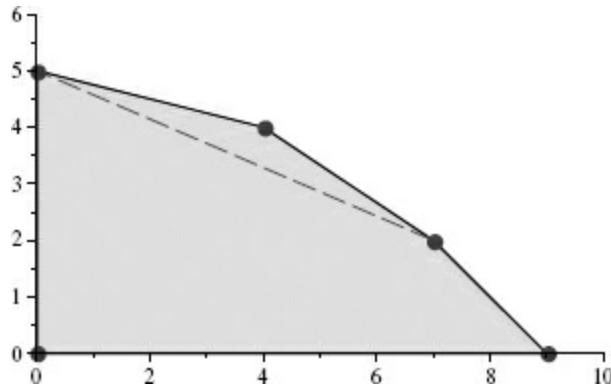
In the polyhedral set S given in (7.1), the following pairs of extreme points are adjacent:

- (0, 0) and (0, 5)
- (0, 5) and (4, 4)
- (4, 4) and (7, 2)
- (7, 2) and (9, 0)
- (9, 0) and (0, 0).

We can see in [Figure 7.5](#) that the extreme points (0, 5) and (7, 2) are not adjacent since the line segment connecting them is not an edge of the feasible region S .

The idea of adjacency is relevant to solving linear programs, and we return to it in Section 7.3.

FIGURE 7.5 Adjacent and nonadjacent extreme points.



When Do Extreme Points Exist in Polyhedrons? A natural question to consider now is whether or not a given polyhedral set contains an extreme point. If it does, we know what it looks like geometrically (and algebraically); unfortunately, this does not guarantee its existence. In fact, as noted earlier, it is fairly easy to construct a polyhedral set that does not contain an extreme point.

■ EXAMPLE 7.8

Consider the polyhedral set $S = \{(x, y) : 1 \leq x + y \leq 3\}$. The set S is the region bounded by two parallel lines in \mathbb{R}^2 and has no extreme points.

For some linear programs, such as those we will commonly see, we can say something more definitive.

Theorem 7.2 *Let $S \subseteq \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x} \geq \mathbf{0}\}$ be a nonempty polyhedral set in the non-negative orthant. Then S has at least one extreme point.*

Proof Suppose S does not have any extreme points. Let $\mathbf{x} \in S$ be the feasible solution that lies on the maximum number $r < n$ linearly independent defining hyperplanes. Since \mathbf{x} is not an extreme point, there exists solutions $\mathbf{x}_1 \neq \mathbf{x}_2$ both in S and $0 < \lambda < 1$ such that $\mathbf{x} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 = \mathbf{x}_2 + \lambda(\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{x}_1 - (1 - \lambda)(\mathbf{x}_1 - \mathbf{x}_2)$. Let $\mathbf{d} = \mathbf{x}_1 - \mathbf{x}_2$ and consider moving from \mathbf{x} along both \mathbf{d} and $-\mathbf{d}$. For small step sizes, both directions yield feasible solutions to S . However, at least one of these directions cannot be unbounded since $S \subseteq \{\mathbf{x} : \mathbf{x} \geq \mathbf{0}\}$. Without loss of generality, we assume that $\delta = \max\{k : \mathbf{x} - k\mathbf{d} \in S\} < \infty$ and consider the feasible solution $\mathbf{y} = \mathbf{x} - \delta\mathbf{d}$. First, note that the linearly independent defining hyperplanes that were active at \mathbf{x} are active at \mathbf{y} , but at least one other linearly independent defining hyperplane is active at \mathbf{y} by construction. Thus, \mathbf{y} lies on more than r linearly independent defining hyperplanes, which is a contradiction.

Corollary 7.1 *A nonempty polyhedral set $S \subseteq \{\mathbf{x} : \mathbf{x} \geq \mathbf{0}\}$ has a finite number of extreme points.*

Let's take a look again at the proof of Theorem 7.2. Recall that we started at the feasible solution \mathbf{x} that was not on n linearly independent defining hyperplanes. We could then find a direction \mathbf{d} that allows us to move from \mathbf{x} in both directions \mathbf{d} or $-\mathbf{d}$. The key idea in the proof was that if we moved from \mathbf{x} in at least one of these directions, then we would hit either a variable bound or a constraint since the polyhedral set was a subset of $\{\mathbf{x} : \mathbf{x} \geq \mathbf{0}\}$; this caused a contradiction. However, if we reexamine this argument, what we find is that if there is no extreme point in a polyhedral set S , then there must be a direction \mathbf{d} and a point $\mathbf{x} \in S$ so that $\mathbf{x} + \lambda\mathbf{d} \in S$ for all λ ; thus, S contains a line. Moreover, the converse is true (see Exercise 7.26).

Theorem 7.3 *Let S be a nonempty polyhedral set. Then S does not contain any extreme points if and only if S contains a line.*

7.2 FUNDAMENTAL THEOREM OF LINEAR PROGRAMMING

Currently, we know we can describe a polyhedral set by its defining hyperplanes, and from this we can describe its extreme points; in Exercise

7.31, we will see that we can similarly describe any unbounded directions of this polyhedral set. Can we describe the set knowing only its extreme points and directions? From our experience with two-variable linear programs, the feasible region appears as the convex hull of the extreme points, except when it was unbounded. We can extend this notion to multidimensional problems through the following **Representation Theorem**.

Theorem 7.4 (Representation Theorem) *Let $S \subseteq \{\mathbf{x} : \mathbf{x} \geq \mathbf{0}\}$ be a polyhedral set and let $V = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ be the extreme points of S . Then $\mathbf{x} \in S$ if and only if*

$$\begin{aligned}\mathbf{x} &= \sum_{i=1}^k \alpha_i \mathbf{v}_i + \mathbf{d} \\ \sum_{i=1}^k \alpha_i &= 1 \\ \alpha_i &\geq 0, \quad i \in \{1, \dots, k\},\end{aligned}$$

where either \mathbf{d} is an unbounded direction of S or $\mathbf{d} = \mathbf{0}$.

Proof See Exercise 7.27 for the case where $S = \{ \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \}$ and S is bounded. A proof of the general case can be found in Bertsimas and Tsitsiklis [14].

In other words, if we know the extreme points and the unbounded directions of a polyhedral set S , then we can generate all solutions in S . This is powerful for theoretical work. One important use of this characterization is the following theorem, called the Fundamental Theorem of Linear Programming.

Theorem 7.5 (Fundamental Theorem of Linear Programming) *Suppose*

$$S \subseteq \{\mathbf{x} : \mathbf{x} \geq \mathbf{0}\}$$

is the nonempty feasible region of some linear program, where $z = \mathbf{c}^T \mathbf{x}$ is its objective function. Then either z attains its optimal value at some extreme point of S or the linear program is unbounded.

Proof Without loss of generality, we shall assume that our linear program wants to maximize z . If S has an unbounded direction \mathbf{d} such that $\mathbf{c}^T \mathbf{d} > 0$, then the linear program is unbounded; hence, we assume $\mathbf{c}^T \mathbf{d} \leq 0$ for any unbounded direction \mathbf{d} . By Theorem 7.4, any $\mathbf{x} \in S$ can be expressed as

$$\begin{aligned}\mathbf{x} &= \sum_{i=1}^k \alpha_i \mathbf{v}_i + \mathbf{d} \\ \sum_{i=1}^k \alpha_i &= 1 \\ \alpha_i &\geq 0, \quad i \in \{1, \dots, k\},\end{aligned}$$

where $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ are the extreme points of S and $\mathbf{c}^T \mathbf{d} \leq 0$. Suppose that \mathbf{v}^* denotes the extreme point whose value $\mathbf{c}^T \mathbf{v}$ is largest. Then,

$$\begin{aligned}\mathbf{c}^T \mathbf{x} &= \mathbf{c}^T \left(\sum_{i=1}^k \alpha_i \mathbf{v}_i + \mathbf{d} \right) \\ &= \sum_{i=1}^k \alpha_i \mathbf{c}^T \mathbf{v}_i + \mathbf{c}^T \mathbf{d} \\ &\leq \left(\sum_{i=1}^k \alpha_i \mathbf{c}^T \mathbf{v}_i \right) \\ &\leq \sum_{i=1}^k \alpha_i \mathbf{c}^T \mathbf{v}^* \\ &= \mathbf{c}^T \mathbf{v}^*.\end{aligned}$$

So for each $\mathbf{x} \in S$, there is an extreme point \mathbf{v}^* whose objective function value is at least as large as that for \mathbf{x} . Hence, an optimal solution to our linear program must be at an extreme point.

The implications of this theorem are huge. We need to consider only extreme points if we are looking for an optimal solution because if the linear program is not unbounded, then at least one optimal solution must exist at an extreme point. We have to be wary of unbounded directions, but at least this gives us a starting point. If we use this logic when we are trying to find an improving direction, we can look for one that points toward an extreme point. We exploit this approach later.

When solving a linear program, we need to consider only extreme points as potential optimal solutions.

It is possible for an optimal solution to a linear program to not be an extreme point; this is not ruled out. However, if an optimal solution exists, and there are extreme points, at least one of the optimal solutions must be at an extreme point. In fact, if there is an optimal solution that is not an extreme point, then there exists an infinite number of optimal

solutions, and at least one of which is an extreme point; that is, there are **multiple solutions** to our linear program.

7.3 LINEAR PROGRAMS IN CANONICAL FORM

When dealing with any mathematical object that can have different representations, such as linear programs, it is often useful to restrict ourselves to one form. If this can be done, then we only need to do our analysis on this canonical form and the results will hold for all representations. In Chapter 2, we saw that linear programs can be written in different equivalent forms. One form that is often useful is to represent all inequalities as equations, all variables to have the same bounds, and in our case be nonnegative, as well as to always maximize our objective function. In other words, we want to construct a *canonical form* of a linear program.¹

Canonical Form of Linear Programs The *canonical form* of a linear program is

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j = b_i, \quad i \in \{1, \dots, m\} \\ & x_j \geq 0, \quad j \in \{1, \dots, n\}. \end{aligned}$$

Another common form has all inequality constraints. This form will be of use to us later, especially in Chapter 9.

Standard Form of Linear Programs The *standard form* of a linear program is

$$\begin{aligned}
\max \quad & \sum_{j=1}^n c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i \in \{1, \dots, m\} \\
& x_j \geq 0, \quad j \in \{1, \dots, n\}.
\end{aligned}$$

Throughout this section, we assume that our linear program is given in canonical form, written here in terms of vectors and matrices:

$$\max \quad \mathbf{c}^T \mathbf{x}$$

$$(7.2a) \quad \text{s.t.}$$

$$(7.2b) \quad A\mathbf{x} = \mathbf{b}$$

$$(7.2c) \quad \mathbf{x} \geq \mathbf{0},$$

where A is an $m \times n$ matrix, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$. To make our studies simpler, we assume that the rank of A is m , implying that all rows are linearly independent. Furthermore, this implies that $m \leq n$, or that there are more variables than constraints in our problem and that no constraint is a linear combination of the others. Note that this is not mathematically restrictive because if either one of these conditions was not satisfied, then we could remove unnecessary constraints until they were met.

■ EXAMPLE 7.9

Consider the following linear program

$$\begin{aligned}
\max \quad & 3x + 8y \\
\text{s.t.} \quad & x + 4y \leq 20 \\
& x + y \leq 9 \\
& 2x + 3y \leq 20 \\
(7.3) \quad & x, y \geq 0.
\end{aligned}$$

Its feasible region is shown in [Figure 7.1](#). Recall that the extreme points (and hence basic feasible solutions) are $(0, 0)$, $(9, 0)$, $(7, 2)$, $(4, 4)$, and $(0, 5)$. In canonical form, this linear program is

$$\begin{aligned}
\max \quad & 3x + 8y \\
\text{s.t.} \quad & \\
& x + 4y + s_1 = 20 \\
& x + y + s_2 = 9 \\
& 2x + 3y + s_3 = 20 \\
(7.4) \quad & x, y, s_1, s_2, s_3 \geq 0.
\end{aligned}$$

In vector-matrix form, this linear program is

$$\begin{aligned}
\max \quad & \begin{bmatrix} x \\ y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \\
\text{s.t.} \quad & \begin{bmatrix} 1 & 4 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 2 & 3 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 9 \\ 20 \end{bmatrix} \\
& \begin{bmatrix} x \\ y \\ s_1 \\ s_2 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.
\end{aligned}$$

Each of the extreme points to the original linear program (7.3) corresponds to the solutions $(0, 0, 20, 9, 20)$, $(9, 0, 11, 0, 2)$, $(7, 2, 5, 0, 0)$, $(4, 4, 0, 1, 0)$, and $(0, 5, 0, 3, 1)$, respectively, of the canonical form (7.4).

Consider the feasible solution $(4, 4, 0, 1, 0)$, for which the active constraints are

$$\begin{aligned}
& x + 4y + s_1 = 20 \\
& x + y + s_2 = 9 \\
& 2x + 3y + s_3 = 20 \\
& s_1 = 0 \\
(7.5) \quad & s_3 = 0.
\end{aligned}$$

These constraints are linearly independent, and so $(4, 4, 0, 1, 0)$ is a basic feasible solution (and hence extreme point). We also note that the columns of the submatrix

$$B = \begin{bmatrix} 1 & 4 & 0 \\ 1 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix}$$

of A corresponding to (nonzero) variables x, y, s_2 are linearly independent and the variables s_1 and s_3 have value 0. We can then rewrite (7.5) in the matrix form

$$\begin{bmatrix} 1 & 4 & 0 \\ 1 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ s_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 9 \\ 20 \end{bmatrix}$$

$$\begin{bmatrix} s_1 \\ s_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We can generalize some of the ideas from this example by letting \mathbf{x} be a basic feasible solution of a linear program in canonical form (7.2). From Theorem 7.1, we know that there are n active linearly independent defining hyperplanes at \mathbf{x} . Since there are already m linearly independent equality constraints the remaining $n - m$ active defining hyperplanes must come from the nonnegativity bounds (7.2c). Suppose that we let the vector \mathbf{x}_N represent the variables of \mathbf{x} that correspond to the $n - m$ active nonnegativity bounds and \mathbf{x}_B as the remaining variables, then we can rewrite the active constraints at \mathbf{x} as

$$B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b}$$

$$\mathbf{x}_N = \mathbf{0}$$

or, equivalently, as

$$(7.6) \quad \begin{bmatrix} B & N \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix},$$

where B is the submatrix of A corresponding to the columns of the variables \mathbf{x}_B and N is the submatrix of A corresponding to the columns of variables \mathbf{x}_N . Since the rows of the matrix in (7.6) are linearly independent, and the matrix is square ($n \times n$), the columns of the matrix are linearly independent. This leads to the following result.

Lemma 7.1 *Suppose $S = \{\mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ is the nonempty feasible region of a linear program in canonical form, and let \mathbf{x} be a basic solution of S . Then \mathbf{x} is a solution to the system of equations $A\mathbf{x} = \mathbf{b}$ and there exists m indices B_1, \dots, B_m such that*

(a) the columns $\mathbf{a}_{B1}, \dots, \mathbf{a}_{Bm}$ of A are linearly independent;

(b) if $j \notin \{B_1, \dots, B_m\}$, then $x_j = 0$.

Thus, basic solutions (feasible or not) of a linear program in canonical form are described not only by the constraints that are active but also by the variables that are potentially nonzero. In fact, the values of the variables given by \mathbf{x}_B can be uniquely found by solving (7.6), giving $\mathbf{x}_B = B^{-1}\mathbf{b}$.

■ EXAMPLE 7.10

Consider the linear program (7.4), and, as per Lemma 7.1, let $\{x, y, s_3\}$ be the variables that are potentially nonzero. We can partition the columns of A and the vector \mathbf{x} to get

$$(7.7) \quad \begin{bmatrix} 1 & 4 & 0 \\ 1 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ s_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 9 \\ 20 \end{bmatrix}$$

$$\begin{bmatrix} s_1 \\ s_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Since $s_1 = s_3 = 0$, we have

$$\begin{aligned} \mathbf{x}_B &= \begin{bmatrix} x \\ y \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 0 \\ 1 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 20 \\ 9 \\ 20 \end{bmatrix} \\ &= \begin{bmatrix} 4 \\ 4 \\ 1 \end{bmatrix}. \end{aligned}$$

This generates the basic (feasible) solution $(x, y, s_1, s_2, s_3) = (4, 4, 0, 1, 0)$.

The following example shows that m linearly independent columns of A do not always generate an extreme point of the feasible region.

■ EXAMPLE 7.11

Consider again Example 7.9. The columns of A corresponding to variables x, y, s_3 are

$$\begin{bmatrix} 1 & 4 & 0 \\ 1 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix}.$$

Although these columns are linearly independent, solving

$$\begin{bmatrix} 1 & 4 & 0 \\ 1 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ s_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 9 \\ 20 \end{bmatrix}$$

yields $x = \frac{16}{3}$, $y = \frac{11}{3}$, and $s_3 = -\frac{5}{3}$. This does not generate an extreme point because the solution $(\frac{16}{3}, \frac{11}{3}, 0, 0, -\frac{5}{3})$ is not even a feasible solution.

What if the solution generated by solving over m linearly independent columns of A was feasible; is this solution an extreme point? The answer, not surprisingly, is yes.

Lemma 7.2 *Given a linear program in canonical form (7.2), suppose that the columns of A are reordered so that the first m columns of A , denoted by $\mathbf{a}_1, \dots, \mathbf{a}_m$, are linearly independent and that*

$$x_1 \mathbf{a}_1 + \dots + x_m \mathbf{a}_m = \mathbf{b},$$

where each $x_i \geq 0$ for $i = 1, 2, \dots, m$. Then the solution

is an extreme point of the feasible region of (7.2).

$$\mathbf{x} = (x_1, x_2, \dots, x_m, 0, 0, \dots, 0)$$

Proof Note that \mathbf{x} is feasible. Assume that \mathbf{x} is not an extreme point. Then there exists feasible solutions \mathbf{u} and \mathbf{v} such that

$$(7.8) \quad \mathbf{x} = \lambda \mathbf{u} + (1 - \lambda) \mathbf{v}, \text{ where } 0 < \lambda < 1.$$

Let us rewrite \mathbf{u} and \mathbf{v} as

$$\begin{aligned} \mathbf{u} &= (u_1, u_2, \dots, u_m, u'_1, u'_2, \dots, u'_{n-m}) \\ \mathbf{v} &= (v_1, v_2, \dots, v_m, v'_1, v'_2, \dots, v'_{n-m}). \end{aligned}$$

Thus, we have

$$\begin{aligned} x_i &= \lambda u_i + (1 - \lambda)v_i, \quad i = 1, 2, \dots, m \\ 0 &= \lambda u'_i + (1 - \lambda)v'_i, \quad i = 1, 2, \dots, n - m. \end{aligned}$$

Since $u'_i, v'_i \geq 0$ for each $i = 1, 2, \dots, n - m$, we have $u'_i = v'_i = 0$. Now, since \mathbf{v} is feasible, we have

$$v_1 \mathbf{a}_1 + v_2 \mathbf{a}_2 + \dots + v_m \mathbf{a}_m = \mathbf{b}.$$

Also, since \mathbf{x} is a feasible solution, we have

$$(x_1 - v_1) \mathbf{a}_1 + \dots + (x_m - v_m) \mathbf{a}_m = \mathbf{0}.$$

Since $\mathbf{a}_1, \dots, \mathbf{a}_m$ are linearly independent, this means that $x_i - v_i = 0$ for all $i = 1, 2, \dots, m$, and hence $\mathbf{x} = \mathbf{v}$, which contradicts (7.8). We conclude that \mathbf{x} is an

extreme point.

Lemmas 7.1 and 7.2 show us that given a linear program in canonical form, any basic feasible solution \mathbf{x} of the feasible region is described by identifying n linearly independent constraints that are active at \mathbf{x} or by identifying m linearly independent columns of A . This greatly simplifies the amount of information needed to describe an extreme point, since there are more variables (n) than constraints (m). If we describe our basic feasible solution by its m columns of A (or equivalently, the corresponding variables), then we should probably give names to these variables.

Basic Variable Given a basic feasible solution \mathbf{x} of a linear program in canonical form, a variable x_k is called *basic* if it is one of the m linearly independent columns of A defining \mathbf{x} .

Nonbasic Variable A variable x_k is *nonbasic* if it is not basic.

Basis The set of basic variables is referred to as the *basis* of \mathbf{x} .

■ EXAMPLE 7.12

Consider the linear program

$$\begin{aligned} \max \quad & 10x + 3y \\ \text{s.t.} \quad & x + y + s_1 = 4 \\ & 5x + 2y + s_2 = 11 \\ & x, y, s_1, s_2 \geq 0. \end{aligned}$$

There are six basic solutions that are tabulated below. Note that only four of them are basic feasible solutions.

x	y	s_1	s_2	Solution Type	Variables	
					Basic	Nonbasic
0	0	4	11	Basic feasible solution	s_1, s_2	x, y
$\frac{11}{5}$	0	$\frac{9}{5}$	0	Basic feasible solution	x, s_1	y, s_2
4	0	0	-9	Basic solution	x, s_2	y, s_1
1	3	0	0	Basic feasible solution	x, y	s_1, s_2
0	$\frac{11}{2}$	$-\frac{3}{2}$	0	Basic solution	y, s_1	x, s_2
0	4	0	3	Basic feasible solution	y, s_2	x, s_1

If we describe basic feasible solutions by only their basic variables, we can partition the constraint matrix A into two submatrices B and N , where B corresponds to the columns of the basic variables and N corresponds to the columns of the nonbasic variables. We also partition the variables \mathbf{x} into their basic components \mathbf{x}_B and non-basic components \mathbf{x}_N . Thus, the partition $[B:N]$ of A denotes the basic solution to which we are referring. In the case of feasibility, we often write the constraints

$$A\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} \geq \mathbf{0}$$

as

$$B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b}$$

$$\mathbf{x}_B \geq \mathbf{0}$$

$$\mathbf{x}_N = \mathbf{0}$$

for the basic feasible solution $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$.

■ EXAMPLE 7.13

Consider the basic feasible solution $(1, 3, 0, 0, 1)$ with basis $\mathcal{B} = \{x, y, s_3\}$ from Example 7.12. We can decompose the constraints

$$\begin{aligned} x + y + s_1 &= 4 \\ 5x + 2y + s_2 &= 11 \\ y + s_3 &= 4 \\ x, y, s_1, s_2, s_3 &\geq 0. \end{aligned}$$

into the following basic and nonbasic parts:

$$\begin{aligned} \begin{bmatrix} 1 & 1 & 0 \\ 5 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ s_3 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} &= \begin{bmatrix} 4 \\ 11 \\ 4 \end{bmatrix} \\ \begin{bmatrix} x \\ y \\ s_3 \end{bmatrix} &\geq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \end{aligned}$$

Where

$$B = \begin{bmatrix} 1 & 1 & 0 \\ 5 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{x}_B = \begin{bmatrix} x \\ y \\ s_3 \end{bmatrix}, \quad \mathbf{x}_N = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}.$$

Note that the columns of B are linearly independent.

Degeneracy It's important to note that Lemmas 7.1 and 7.2 do **not** tell us the number of positive components equals the number of constraints. The following example illustrates this fact.

■ EXAMPLE 7.14

Let us change the problem in Example 7.9 to

$$\begin{aligned} \max \quad & 8x + 3y \\ \text{s.t.} \quad & \\ & x + 4y + s_1 = 20 \\ & x + y + s_2 = 9 \\ & 2x + 3y + s_3 = 20 \\ & x + 2y + s_4 = 12 \\ (7.9) \quad & x, y, s_1, s_2, s_3, s_4 \geq 0. \end{aligned}$$

Consider the feasible solution $(4, 4, 0, 1, 0, 0)$. Note that seven (and not $n = 6$) constraints and bounds are active at the solution $(4, 4, 0, 1, 0, 0)$, making it degenerate. Also, note that there are three (and not $m = 4$) positive values.

Finally, this solution is a basic feasible solution corresponding to a basis $\mathcal{B} = \{x, y, s_2, s_4\}$

Lemma 7.3 *A basic feasible solution \mathbf{x} is degenerate if there are less than m positive coordinates, where m is the number of constraints.*

Thus, degenerate basic feasible solutions are defined as having either too many constraints active at it or too few positive components. This notion causes some difficulties in equating the notions of extreme points (the geometric description) and basic feasible solutions (an algebraic description).

■ EXAMPLE 7.15

Consider, again, the linear program (7.9) and the basic feasible solution $(4, 4, 0, 1, 0, 0)$. Notice that the bases $\mathcal{B}_1 = \{x, y, s_2, s_4\}$ and $\mathcal{B}_2 = \{x, y, s_2, s_3\}$ both describe this basic feasible solution.

The above examples illustrate the fundamental relationship between the basic feasible solutions and the bases that describe it.

Theorem 7.6 *Every basic feasible solution corresponds to at least one basis. If the basic feasible solution is not degenerate, there is exactly one basis that*

describes it.

Because basic (feasible) solutions can have more than one basis that describes it, we will denote a basic feasible solution by its basis. Thus, if we have a degenerate basic feasible solution, we will say that each set of basic variables describing it is a different basic feasible solution.

■ EXAMPLE 7.16

The linear program

$$\begin{aligned} \max \quad & 10x + 3y \\ \text{s.t.} \quad & x + y + s_1 = 4 \\ & 5x + 2y + s_2 = 11 \\ & y + s_3 = 4 \\ (7.10) \quad & x, y, s_1, s_2, s_3 \geq 0 \end{aligned}$$

has nine basic solutions (seven distinct solutions with one being degenerate) given in the following table.

x	y	s ₁	s ₂	s ₃	Solution Type	Variables	
						Basic	Nonbasic
0	0	4	11	4	Basic feasible solution	s ₁ , s ₂ , s ₃	x, y
$\frac{11}{5}$	0	$\frac{9}{5}$	0	4	Basic feasible solution	x, s ₁ , s ₃	y, s ₂
4	0	0	-9	4	Basic solution	x, s ₂ , s ₃	y, s ₁
1	3	0	0	1	Basic feasible solution	x, y, s ₃	s ₁ , s ₂
0	$\frac{11}{2}$	$-\frac{3}{2}$	0	$-\frac{3}{2}$	Basic solution	y, s ₁ , s ₃	x, s ₂
0	4	0	3	0	Basic feasible solution	y, s ₂ , s ₃	x, s ₁
0	4	0	3	0	Basic feasible solution	y, s ₁ , s ₂	x, s ₃
0	4	0	3	0	Basic feasible solution	x, y, s ₂	s ₁ , s ₃
$\frac{3}{5}$	4	$-\frac{3}{5}$	0	0	Basic solution	x, y, s ₁	s ₂ , s ₃ .

The 10th combination of three columns, corresponding to variables x, s₁, s₂, is not linearly independent.

Adjacency In Examples 7.12 and 7.16, there were pairs of basic (feasible) solutions whose basic variable lists differ by at most one variable. This

sounds like the idea of adjacency, doesn't it?

Adjacent Basic Solutions Two basic solutions are *adjacent* if their bases differ by exactly one element.

■ EXAMPLE 7.17

In Example 7.16, the basic feasible solutions $(0, 0, 4, 11, 4)$ and $(\frac{11}{5}, 0, \frac{9}{5}, 0, 4)$ are adjacent since their bases are $\mathcal{B}_1 = \{s_1, s_2, s_3\}$ and $\mathcal{B}_2 = \{x, s_1, s_3\}$, respectively. Also, the basic feasible solutions $(0, 0, 4, 11, 4)$ and $(0, 4, 0, 3, 0)$ are adjacent (bases $\{s_1, s_2, s_3\}$ and $\{y, s_2, s_3\}$), as well as the basic feasible solutions $(0, 4, 0, 3, 0)$ and $(0, 4, 0, 3, 0)$ (bases $\{y, s_2, s_3\}$ and $\{x, y, s_2\}$).

In Example 7.16, two basic feasible solutions may be adjacent even if they correspond to the same extreme point. This occurs only when the extreme point is degenerate.

How does the notion of adjacent basic feasible solutions relate to that of adjacent extreme points given in Section 7.1? Assume for simplicity that none of our extreme points are degenerate. We saw that two different extreme points are adjacent if they both lie on $n - 1$ linearly independent defining hyperplanes. In canonical form, each extreme point lies on the m equality constraints and $n - m$ variable bounds $x_k = 0$, which means that for two adjacent extreme points, there exists a variable x_j that has value 0 at one of the extreme points and is positive at the other. Thus, x_j is basic in one solution and nonbasic in the other, but this is the definition of adjacent basic feasible solutions! Hence, as we noted before, the geometric notion of adjacency (with extreme points) and the algebraic notion (with basic feasible solutions) mimic each other.

■ EXAMPLE 7.18

Recall the linear program (7.10) and consider the adjacent basic feasible solutions $(1, 3, 0, 0, 1)$ with basis $\{x, y, s_3\}$ and $(\frac{11}{5}, 0, \frac{9}{5}, 0, 4)$ with basis $\{x, s_1, s_3\}$. Each of these basic feasible solutions is extreme point of our feasible region and lies on the defining hyperplanes

$$\begin{aligned}
 x + y + s_1 &= 4 \\
 5x + 2y + s_2 &= 11 \\
 y + s_3 &= 4 \\
 (7.11) \quad s_2 &= 0
 \end{aligned}$$

Since these hyperplanes are linearly independent, the two extreme points are adjacent. To find the edge connecting them, we shall base our line at $(1, 3, 0, 0, 1)$. Since $s_1 = 0$ at this solution and is positive at the other, we can find the edge by adding the equation

$$s_1 = t$$

to (7.11), indicating that t is our line (degree of freedom) parameter. Solving this system yields the line

$$\begin{bmatrix} 1 \\ 3 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ -5 \\ 3 \\ 0 \\ 5 \end{bmatrix} t$$

that generates a feasible solution for $t \in [0, \frac{3}{5}]$, where $t = \frac{3}{5}$ is the largest value of t for which the solution remains nonnegative. It is easy to see that $\mathbf{d} = (2, -5, 3, 0, 5)$ is a feasible direction from $(1, 3, 0, 0, 1)$.

We should be careful in how we interpret the above example, in which we identified an edge between two nondegenerate basic feasible solutions and showed that we can move along this edge a positive amount and remain feasible. If, however, we have two adjacent degenerate basic feasible solutions, this may not be the case (see Exercise 7.38).

Summary

We have characterized both algebraically and geometrically the extreme points of polyhedral sets. The Fundamental Theorem of Linear Programming states that if an optimal solution to a linear program exists, we need to examine only the extreme points. We also saw the importance of writing linear programs in canonical form—since every linear program can be transformed into canonical form, we need to explore only algorithms designed to handle this form. In addition, basic feasible solutions are characterized by their basic variables, which creates a compact way to distinguish basic feasible solutions. In Chapter 8, we will use this knowledge

to determine appropriate improving search directions for our improving search algorithm so that we can solve linear programs.

EXERCISES

7.1 Determine the extreme points of the following convex region:

$$\begin{aligned} -2x_1 + x_2 &\leq 2 \\ x_1 + x_2 &\leq 6 \\ x_1 &\leq 4 \\ x_1, x_2 &\geq 0. \end{aligned}$$

7.2 Determine the extreme points of the following convex region:

$$\begin{aligned} -x_1 + x_2 &\leq 4 \\ x_1 - x_2 &\leq 10 \\ x_1 + x_2 &\leq 12 \\ x_1, x_2 &\geq 0. \end{aligned}$$

7.3 Determine the extreme points of the following polyhedral set. For each extreme point, identify the linearly independent constraints defining it.

$$\begin{aligned} x_1 + x_2 + x_3 &\leq 5 \\ -x_1 + x_2 + 2x_3 &\leq 6 \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

7.4 Identify all extreme points of the following polyhedral set:

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

7.5 Given the solutions $P = \{(1, 2), (4, 3), (5, 7), (0, 7)\}$, what linear inequalities describe the convex hull of P ? What are its extreme points?

7.6 Given the solutions $P = \{(1, 1), (2, 5), (3, 3), (4, 6), (5, 2), (6, 3)\}$, what linear inequalities describe the convex hull of P ? What are its extreme points?

7.7 For the feasible region found in Exercise 7.5,

(a) For each extreme point, find an (nonconstant) objective function that makes it uniquely optimal.

(b) For each pair of adjacent extreme points, find an (nonconstant)

objective function that makes both optimal.

7.8 For the feasible region found in Exercise 7.6,

(a) For each extreme point, find an (nonconstant) objective function that makes it uniquely optimal.

(b) For each pair of adjacent extreme points, find an (nonconstant) objective function that makes both optimal.

7.9 Generalizing the previous two exercises, suppose you are given any polytope P with extreme points $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ in \mathbb{R}^n . For any extreme point \mathbf{v}_j , determine a method for obtaining an (nonconstant) objective function that makes it uniquely optimal.

7.10 Show that the solution $(2, 1, 1)$ is a degenerate solution to consider the polyhedral set defined by

$$\begin{aligned}x & - 2z \leq 0 \\x + 2y & \leq 4 \\x & + 2z \leq 4 \\x - 2y & \leq 0 \\x, y, z & \geq 0.\end{aligned}$$

(a) Show that the solution $(2, 1, 1)$ is a degenerate basic feasible solution.

(b) Show that none of these constraints is redundant.

7.11 In 1907, C. Carathéodory proved that, given a collection S of at least $n + 1$ points in \mathbb{R}^2 , a point \mathbf{z} is in the convex hull of S if and only if it is in the convex hull of at most $n + 1$ such solutions from S . Using the points given in Exercise 7.6, where $n = 2$, show that the solution $(3, 2)$ can be written as the convex combination of at most 3 of the extreme points.

7.12 Suppose in Problem 7.11 that $S = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ is the set of extreme points of a polyhedral set, where $k > n$. Prove Carathéodory's theorem for this set. (*Hint:* Set up an appropriate linear program.)

7.13 Consider the following linear program:

$$\begin{aligned}\max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0},\end{aligned}$$

where the feasible region is bounded. Let $V = \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ be the extreme points of the feasible region. Knowing only the extreme points of the feasible set, construct a linear program whose solution is the same as the solution to the original linear program above.

7.14 Construct the canonical form of the following linear programs.

(a)

$$\begin{aligned} & \max \quad 3x + 2y \\ \text{s.t.} \quad & 2x - y \leq 6 \\ & 2x + y \leq 10 \\ & x, y \geq 0. \end{aligned}$$

(b)

$$\begin{aligned} & \max \quad x + y \\ \text{s.t.} \quad & -2x + y \leq 0 \\ & x - 2y \leq 0 \\ & x + y \leq 9 \\ & x, y \geq 0. \end{aligned}$$

(c)

$$\begin{aligned} & \max \quad 4x_1 + 2x_2 + 7x_3 \\ \text{s.t.} \quad & 2x_1 - x_2 + 4x_3 \leq 18 \\ & 4x_1 + 2x_2 + 5x_3 \geq 10 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

(d)

$$\begin{aligned} & \min \quad 2x_1 - x_2 + 3x_3 \\ \text{s.t.} \quad & x_1 - x_2 + 4x_3 \leq 8 \\ & 2x_1 + 2x_2 - 5x_3 = 4 \\ & x_1 + x_3 \geq 6 \\ & x_1, x_3 \geq 0, x_2 \leq 0. \end{aligned}$$

(e)

$$\begin{aligned}
& \max && x_1 + 2x_2 \\
& \text{s.t.} && \\
& & 4x_1 - 2x_2 + 3x_3 \leq 13 \\
& & 5x_1 - 2x_2 & \geq 10 \\
& & x_1, x_2 \geq 0.
\end{aligned}$$

7.15 For each linear program in Exercise 7.14, write the canonical form in matrix notation.

7.16 Consider the system of equations $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 2 & 3 & 1 & 0 & 0 \\ -1 & 1 & 0 & 2 & 1 \\ 0 & 6 & 1 & 0 & 3 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix}.$$

Determine which of the following are basic solutions to the system.

- (a)** $(1, 0, -1, 1, 0)$.
- (b)** $(0, 2, -5, 0, -1)$.
- (c)** $(0, 0, 1, 0, 1)$.

7.17 Suppose the canonical form of a linear programming problem is given by the constraint matrix A and right-hand-side vector \mathbf{b} , where

$$A = \begin{bmatrix} 3 & 0 & 1 & 1 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 4 & 0 & 3 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 5 \\ 3 \\ 6 \end{bmatrix}.$$

Determine (and justify) which of the following solutions is

- (i) a feasible solution to the linear programming problem.
- (ii) an extreme point of the feasible region.
- (iii) a basic solution.
- (iv) a basic feasible solution.

For each basic feasible solution, list the basic variables.

- (a)** $(0, 3, 0, 5, 6)$.
- (b)** $(0, 3, 5, 0, -9)$.
- (c)** $(\frac{3}{2}, 0, 0, \frac{1}{2}, 0)$.
- (d)** $(\frac{1}{2}, 1, 1, 0, 2)$.
- (e)** $(1, 1, \frac{1}{2}, \frac{3}{2}, \frac{1}{2})$.

7.18 Given the linear program in Exercise 7.1,

- (a) Add appropriate slack variables s_1, s_2, s_3 to place the constraints in canonical form.
- (b) For each extreme point found, determine the corresponding extreme point to the feasible region of the canonical form version.
- (c) For each extreme point found, determine the corresponding basic variables.

7.19 Given the linear program in Exercise 7.4,

- (a) Add appropriate slack variables s_1, s_2 to place the constraints in canonical form.
- (b) Determine and justify whether the following set of variables generate a basic solution.

- (i) $\{x_1, x_2\}$.
- (ii) $\{x_1, s_2\}$.
- (iii) $\{s_1\}$.
- (iv) $\{x_1, s_1, s_2\}$.
- (v) $\{s_1, s_2\}$.

- (c) For each of the sets in part (b) that does generate a basic solution, determine that solution.

7.20 Given the polyhedral set in Exercise 7.3,

- (a) Write the polyhedral set in canonical form.
- (b) Determine the basic feasible solutions.
- (c) How do the answers in (b) relate to the answers found in Exercise 7.3? In particular, what is the relationship between the linearly independent constraints and basic feasible solutions?

7.21 For each linear program in Exercise 7.14, identify the basic and nonbasic variables for all basic feasible solutions.

7.22 For each set of basic feasible solutions to a linear programming problem found in Exercise 7.21, identify the pairs of adjacent solutions.

7.23 What is the largest number of extreme points that a linear program can have? Assume that the canonical form has m constraints and n variables.

7.24 Suppose you are given a linear program in standard form

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j \leq b_i \quad i \in \{1, \dots, m\} \\ & x_j \geq 0 \quad j \in \{1, \dots, n\} \end{aligned}$$

where each $b_i \geq 0$. Show that the origin $\mathbf{0} = (0, 0, \dots, 0)$ is an extreme point of the feasible region.

7.25 Suppose that, in our initial linear program, we have a variable x that is unrestricted in sign. When converting the linear program to canonical form, we replace it with $x = x^+ - x^-$. Show that, in every basic feasible solution, at least one of the variables x^+, x^- has a value of 0.

7.26 We shall prove part of Theorem 7.3. Suppose S is a nonempty polyhedral written as $S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \geq \mathbf{b}\}$ and suppose $\mathbf{v} \in S$ is an extreme point of S . Show that S does not contain a line $\mathbf{x} + \lambda \mathbf{d}$, where $\mathbf{d} \neq \mathbf{0}$. (*Hint:* Assume such a line exists and show that \mathbf{d} must equal $\mathbf{0}$.)

7.27 In this problem, we shall step through a proof of Theorem 7.4 for the case where

$$S = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

and S is bounded, with A being an $m \times n$ matrix with rank m . Let $V = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ be the set of extreme points of S . Suppose $\mathbf{x} \in S$, but $\mathbf{x} \notin V$.

(a) Show that there exists a direction $\mathbf{p} \neq \mathbf{0}$ such that

$$\begin{aligned} A\mathbf{p} &= \mathbf{0} \\ p_j &= 0 \quad \text{if } x_j = 0. \end{aligned}$$

This then implies that, for ϵ small in magnitude, we have

$$\begin{aligned} A(\mathbf{x} + \epsilon \mathbf{p}) &= \mathbf{b} \\ \mathbf{x} + \epsilon \mathbf{p} &\geq \mathbf{0} \\ x_j + \epsilon p_j &= 0 \quad \text{if } x_j = 0. \end{aligned}$$

(b) Show that there exists $\epsilon_1 > 0$ and $\epsilon_2 < 0$ such that the solutions

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{x} + \epsilon_1 \mathbf{p} \\ \mathbf{y}_2 &= \mathbf{x} + \epsilon_2 \mathbf{p} \end{aligned}$$

are both in S and have at least one more zero element than \mathbf{x} and that \mathbf{x} is a convex combination of \mathbf{y}_1 and \mathbf{y}_2 .

(c) Show that if \mathbf{y}_1 and \mathbf{y}_2 are each convex combinations of the extreme points in V , then so is any convex combination of \mathbf{y}_1 and \mathbf{y}_2 .

(d) Finish the proof of Theorem 7.4 by using induction.

7.28 Suppose that a linear program with a bounded feasible region has K optimal extreme points $\mathbf{v}_1, \dots, \mathbf{v}_K$. Show that a solution \mathbf{x} is optimal for this linear program if and only if it is a convex combination of $\mathbf{v}_1, \dots, \mathbf{v}_K$.

7.29 Prove that for a convex set S if \mathbf{d} is an unbounded direction of S , then $k\mathbf{d}$ is also an unbounded direction of S , provided $k > 0$.

7.30 Prove that if $\mathbf{d}_1, \dots, \mathbf{d}_k$ are unbounded directions of a convex set S , then so is

$$\mathbf{d} = \sum_{j=1}^k \beta_j \mathbf{d}_j,$$

provided each $\beta_j \geq 0$. An unbounded direction \mathbf{d} of S that cannot be written as the nonnegative sum of at least two unbounded directions of S is called an **extreme direction** of S .

7.31 Consider the unbounded polyhedral set $S = \{\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$.

(a) Show that the set of unbounded directions is $D = \{\mathbf{d} : A\mathbf{d} \leq \mathbf{0}, \mathbf{d} \geq \mathbf{0}, \mathbf{d} \neq \mathbf{0}\}$.

(b) Show that D can be rewritten as

$$D = \{\mathbf{d} : A\mathbf{d} \leq \mathbf{0}, \mathbf{d} \geq \mathbf{0}, \mathbf{e}^T \mathbf{d} = 1\},$$

where $\mathbf{e} = (1, 1, \dots, 1)$ is the vector of all ones. The extreme points of D are therefore the extreme directions of S .

7.32 Consider the polyhedral set S defined by the constraints

$$x - 3y \leq -2$$

$$x - y \leq 2$$

$$x - y \leq 8$$

$$y \geq 2$$

$$x, y \geq 0.$$

(a) Characterize the set D of unbounded directions of S .

(b) Identify the extreme directions of S .

7.33 Consider the linear program

$$\begin{aligned} \max \quad & c_x x + c_y y \\ \text{s.t.} \quad & x - y \leq 2 \\ & 2x - y \geq -4 \\ (7.12) \quad & x, y \geq 0. \end{aligned}$$

(a) Identify every extreme point of the feasible region.

(b) Identify every extreme direction of the feasible region.

(c) Characterize the set of values (c_x, c_y) for which there is a finite optimal solution.

(d) Characterize the set of values (c_x, c_y) for which there is an unbounded solution.

7.34 Consider the feasible region of the linear program (7.12). For each of the following feasible solutions, use the Representation Theorem to write it as a convex combination of extreme points and some unbounded direction.

(a) $(1, 2)$.

(b) $(2, 1)$.

(c) $(6, 3)$.

7.35 Consider the linear program

$$\begin{aligned} \min \quad & c_x x + c_y y \\ \text{s.t.} \quad & 3x - y \geq -3 \\ & x - 2y \leq 4 \\ & x - 3y \leq 2 \\ (7.13) \quad & x, y \geq 0. \end{aligned}$$

(a) Identify every extreme point of the feasible region.

(b) Identify every extreme direction of the feasible region.

(c) Characterize the set of values (c_x, c_y) for which there is a finite optimal solution.

(d) Characterize the set of values (c_x, c_y) for which there is an unbounded solution.

7.36 Consider the feasible region of the linear program (7.13). For each of the following feasible solutions, use the Representation Theorem to write it as a convex combination of extreme points and some unbounded direction.

(a) (5, 1).

(b) (2, 2).

(c) (6, 6).

7.37 Consider the linear program (7.10) and the basic feasible solutions with bases $\mathcal{B}_1 = \{x, y, s_3\}$ and $\mathcal{B}_2 = \{y, s_2, s_3\}$.

(a) Find the defining hyperplanes that are active at both basic feasible solutions.

(b) To identify the edge between them, suppose we set $s_2 = t$ for \mathcal{B}_1 (to make s_2 basic in the new solution). Find the direction \mathbf{d} for this edge.

(c) Calculate the maximum step size we can take in this direction from the first solution if we want to remain feasible.

7.38 Consider the linear program (7.10) and consider the adjacent basic feasible solutions with bases $\mathcal{B}_1 = \{x, y, s_2\}$ and $\mathcal{B}_2 = \{y, s_1, s_2\}$.

(a) Find the defining hyperplanes that are active at both basic feasible solutions.

(b) To identify the “edge” between them, suppose we set $s_1 = t$ for \mathcal{B}_1 (to make s_1 basic in the new solution). Find the direction \mathbf{d} for this edge and show that the maximum step size we can take in this direction from the first solution if we want to remain feasible is 0.

(c) Why is this maximum step size 0?

¹Our definitions of canonical and standard forms of linear programs may differ from other texts. There is no consensus regarding these definitions.

CHAPTER 8

SOLVING LINEAR PROGRAMS: SIMPLEX METHOD

The previous two chapters introduced some basic concepts in optimization, such as improving search methods and the geometry and algebra of linear programs. We now combine these topics to produce an algorithm for solving linear programs called the **Simplex Method**. The simplex method is the most widely used optimization algorithm, solving daily thousands of real-world problems for business, industry, academe, and other scientific outlets. Various implementations routinely solve problems with millions of constraints and tens of millions of variables. In this chapter, we learn the fundamentals of the simplex method, how to implement it efficiently, and how it deals with various “problem areas.”

8.1 SIMPLEX METHOD

The simplex method is a specialized version of the general improving search algorithm (Algorithm 6.2) that is designed to take advantage of the properties of linear programs. Like many improving search algorithms, the simplex method moves from feasible solution to feasible solution, improving the objective function value at each step; however, the simplex method examines only certain types of solutions.

Because linear programs can appear in many equivalent forms, we concentrate on solving only linear programs expressed in the *canonical form*

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \\ & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{8.1}$$

with m constraints and n variables. We assume the rank of a is m and that $m \leq n$. From the fundamental theorem of linear programming (Theorem 7.5), if a linear program has an optimal solution then at least one such solution exists at a basic feasible solution, which implies that we need to consider only basic feasible solutions. For a linear program in canonical form (8.1), a basic feasible solution is a solution for which the variables are partitioned into m basic variables and $n - m$ nonbasic variables, the columns of a associated with the basic variables (denoted by the matrix B) are linearly independent, the values of the nonbasic variables \mathbf{x}_N are zero, and the values of the basic variables \mathbf{x}_B uniquely solve $B\mathbf{x}_B = \mathbf{b}$ and are nonnegative.

We also use a matrix representation of the problem. Suppose our current basic feasible solution is \mathbf{x}^0 , where \mathcal{B} is the set of basic variables and \mathcal{N} is the set of nonbasic variables. We can rewrite our linear program to reflect this solution by partitioning the constraint matrix a , the vector of variables \mathbf{x} , and the objective coefficients \mathbf{c} into their basic and nonbasic components,

$$A = [B : N], \quad \mathbf{c} = \begin{bmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix},$$

where \mathbf{c}_B (\mathbf{c}_N) is the vector of objective coefficients for the basic (nonbasic) variables. This leads to the reformulation

$$\begin{aligned} \max \quad & \mathbf{c}_B^T \mathbf{x}_B + \mathbf{c}_N^T \mathbf{x}_N \\ \text{s.t.} \quad & B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b} \\ & \mathbf{x}_B \geq \mathbf{0} \\ & \mathbf{x}_N = \mathbf{0}. \end{aligned}$$

■ EXAMPLE 8.1

Consider the following linear program:

$$\begin{aligned} \max \quad & 13x + 5y \\ \text{s.t.} \quad & 4x + y \leq 24 \\ & x + 3y \leq 24 \\ & 3x + 2y \leq 23 \\ & x, y \geq 0. \end{aligned}$$

In Chapter 1, we solved this problem using a graphical technique and found

that the optimal solution was (5, 4). In canonical form, we have

$$\begin{aligned} \max \quad & 13x + 5y \\ \text{s.t.} \quad & \\ 4x + y + s_1 &= 24 \\ x + 3y + s_2 &= 24 \\ 3x + 2y + s_3 &= 23 \\ (8.2) \quad & x, y, s_1, s_2, s_3 \geq 0. \end{aligned}$$

The basic feasible solution (0, 0, 24, 24, 23) corresponds to the origin in the previous form and has the basis $\mathcal{B} = \{s_1, s_2, s_3\}$. In matrix form, we have

$$\begin{aligned} B &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad N = \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 3 & 2 \end{bmatrix}, \\ \mathbf{c}_B &= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{c}_N = \begin{bmatrix} 13 \\ 5 \end{bmatrix}, \\ \mathbf{x}_B &= \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}, \quad \mathbf{x}_N = \begin{bmatrix} x \\ y \end{bmatrix}. \end{aligned}$$

Our partitioned linear program is

$$\begin{aligned} \max \quad & [0 \ 0 \ 0] \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} + [13 \ 5] \begin{bmatrix} x \\ y \end{bmatrix} \\ \text{s.t.} \quad & \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} + \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 24 \\ 24 \\ 23 \end{bmatrix} \\ \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned}$$

In order to create an improving search algorithm, we need to (1) obtain a starting solution, (2) construct an improving feasible direction, and (3) identify a step length. We look at each of these notions individually.

Initial Solutions

Because the simplex method will examine only basic feasible solutions, we

need to find an initial one.

■ EXAMPLE 8.2

In Example 8.1, we saw that the basic feasible solution $(0, 0, 24, 24, 23)$ corresponded to the origin in the original formulation of the problem. The basis $\mathcal{B} = \{s_1, s_2, s_3\}$ then consists of only the added slack variables.

Example 8.2 illustrates a nice property concerning linear programs in a standard form.

If $\mathbf{b} \geq \mathbf{0}$ in

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

then the origin is a basic feasible solution. Thus, in canonical form we have

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} + I\mathbf{s} = \mathbf{b} \\ & \mathbf{x}, \mathbf{s} \geq \mathbf{0}, \end{aligned}$$

and the solution $(\mathbf{x}, \mathbf{s}) = (\mathbf{0}, \mathbf{b})$ is a basic feasible solution with basic variables \mathbf{s} , and hence is a possible starting point for the simplex method.

While not every linear program has easy-to-find basic feasible solutions, for now we assume an initial starting solution is known. In Section 8.4, we discuss how to obtain initial basic feasible solutions when they are not obvious.

Finding Feasible Directions

Our next goal is to determine an improving feasible direction. Since we're going to consider only basic feasible solutions in our algorithm, our direction must point toward a basic feasible solution. One approach is to move toward an adjacent basic feasible solution. Recall that two basic feasible solutions are

adjacent if the basis for each solution differs by exactly one variable. Hence, if we are at a basic feasible solution \mathbf{x}_0 with basis \mathcal{B}_0 and want to move toward an adjacent basic feasible solution \mathbf{x}_1 with basis \mathcal{B}_1 , then there is one nonbasic variable of \mathbf{x}_0 that is in \mathcal{B}_1 and one nonbasic variable in \mathbf{x}_1 that is in \mathcal{B}_0 . We saw an example of this in Chapter 7 (Example 7.18) where we found the edge joining two basic feasible solutions.

Let \mathbf{d} denote the direction we wish to determine. If we want to consider the basic feasible solution where nonbasic variable x_k is to become basic, our direction \mathbf{d} must have $d_k > 0$ since we are (hopefully) increasing the value of variable x_k from zero; this is akin to Example 7.18 for which the active constraint $x_k = 0$ will no longer be active in the new solution. We can assume that our direction \mathbf{d} is scaled so that $d_k = 1$, and since all other nonbasic variables x_j remain nonbasic (and hence their constraints remain active), we have $d_j = 0$ for all other nonbasic variables x_j .

Remembering that any direction we choose must also be a feasible direction, we compute the components d_i for all basic variables x_i to preserve feasibility. Recall from Chapter 6 that if a constraint

$$\sum_{j=1}^n a_j x_j \begin{cases} \leq \\ \geq \\ = \end{cases} b$$

is active at the current solution, then any feasible direction \mathbf{d} must satisfy

$$\sum_{j=1}^n a_j d_j \begin{cases} \leq \\ \geq \\ = \end{cases} 0.$$

Since our linear program is in canonical form with only equality constraints, $a\mathbf{x} = \mathbf{b}$, every feasible direction \mathbf{d} satisfies

(8.3) $A\mathbf{d} = \mathbf{0}$.

Simplex Direction A *simplex direction* \mathbf{d}^k corresponding to the nonbasic variable x_k is a direction vector where (1) $d_k^k = 1$, (2) $d_j^k = 0$ for all other nonbasic variables $x_j \neq x_k$, and (3) each d_i^k component corresponding to the basic variables x_i is (uniquely) determined by solving (8.3).

■ EXAMPLE 8.3

Consider the linear program (8.2) given in Example 8.1. Our initial basic feasible solution is $(0, 0, 24, 24, 23)$ with basis $\mathcal{B} = \{s_1, s_2, s_3\}$. The form of the simplex direction \mathbf{d}^x for the nonbasic variable x is

$$\mathbf{d}^x = \begin{bmatrix} 1 \\ 0 \\ d_{s_1} \\ d_{s_2} \\ d_{s_3} \end{bmatrix}.$$

For \mathbf{d}^x to be feasible, it has to satisfy (8.3), that is, the system

$$\begin{aligned} 4 + d_{s_1} &= 0 \\ 1 + d_{s_2} &= 0 \\ 3 + d_{s_3} &= 0, \end{aligned}$$

generating the direction $\mathbf{d}^x = (1, 0, -4, -1, -3)$. This implies, for example, that every unit increase in the x -variable decreases the variable s_1 by 4.

The simplex direction \mathbf{d}^y for nonbasic variable y is of the form

$$\mathbf{d}^y = \begin{bmatrix} 0 \\ 1 \\ d_{s_1} \\ d_{s_2} \\ d_{s_3} \end{bmatrix},$$

where d_{s_1}, d_{s_2} , and d_{s_3} are the solutions to the system of equations

$$\begin{aligned} 1 + d_{s_1} &= 0 \\ 3 + d_{s_2} &= 0 \\ 2 + d_{s_3} &= 0. \end{aligned}$$

Hence, the simplex direction \mathbf{d}^y is $\mathbf{d}^y = (0, 1, -1, -3, -2)$.

When we computed each simplex direction in Example 8.3, we constructed a system of equations with a unique solution. This is always the case because the columns of A corresponding to the basic variables are linearly independent. Since these columns correspond to the only variables in the system of [equations \(8.3\)](#), there is always a unique solution, and hence a unique direction.

Now consider this calculation in matrix form. The simplex direction \mathbf{d}^k for

the nonbasic variable x_k satisfies

$$\begin{aligned} A\mathbf{d}^k &= \mathbf{0} \\ d_k^k &= 1 \\ d_j^k &= 0, \quad j \in \mathcal{N} - \{k\}. \end{aligned}$$

Thus, we solve a system of equations in the basic variables, where the right-hand side is always the negative of the column \mathbf{a}_k in A corresponding to variable x_k . This system of equations can be written as

$$(8.4) \quad B\mathbf{d}_B^k = -\mathbf{a}_k,$$

where \mathbf{d}_B^k is the vector corresponding to the basic variable components of \mathbf{d}^k .

■ EXAMPLE 8.4

Consider the computation done to compute the simplex direction \mathbf{d}^x in Example 8.3. The system of equations needed to be solved was

$$\begin{aligned} 4 + d_{s_1} &= 0 \\ 1 + d_{s_2} &= 0 \\ 3 + d_{s_3} &= 0. \end{aligned}$$

This is equivalent to the system of equations

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_{s_1} \\ d_{s_2} \\ d_{s_3} \end{bmatrix} = - \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix},$$

which is of the form (8.4).

One way to solve (8.4) is to compute

$$(8.5) \quad \mathbf{d}_B^k = -B^{-1}\mathbf{a}_k.$$

Why don't we do this? While the solutions would be the same, it is computationally quicker and more accurate to solve (8.4). In general, anytime a theoretical answer involves the inverse of a matrix, we solve the related system of equations instead of computing the inverse directly.

Determining Improving Directions

Now that we've found all the possible feasible directions that will move us to an adjacent basic feasible solution, we determine whether any of these are improving directions. Recall that if we are maximizing an objective function f

(\mathbf{x}), a direction \mathbf{d} is improving if

$$\nabla f(\mathbf{x})^T \mathbf{d} > 0.$$

Since our function is linear ($f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$), the gradient is $\nabla f(\mathbf{x}) = \mathbf{c}$. So, for each simplex direction \mathbf{d}^k we compute $\mathbf{c}^T \mathbf{d}^k$.

Reduced Cost The *reduced cost* \bar{c}_k associated with the nonbasic variable x_k is

$$\bar{c}_k = \mathbf{c}^T \mathbf{d}^k = c_k + \sum_{i \in \mathcal{B}} c_i d_i^k,$$

where \mathbf{d}^k is the simplex direction associated with variable x_k and \mathcal{B} is the set of basic variables.

Vector of Reduced Costs The *vector of reduced costs* $\bar{\mathbf{c}}_N$ is the vector of all reduced costs \bar{c}_k for nonbasic variables x_k .

The term “reduced cost” is used since the simplex direction \mathbf{d}^k indicates how the value of each variable changes per unit increase in the value of nonbasic variable x_k . Thus, the value \bar{c}_k yields the change in objective value (per unit increase in x_k) if we move from the current basic feasible solution in the direction \mathbf{d}^k .

The simplex direction \mathbf{d}^k associated with nonbasic variable x_k is an improving direction if

- (1) $\bar{c}_k > 0$ for a maximization problem;
- (2) $\bar{c}_k < 0$ for a minimization problem.

■ EXAMPLE 8.5

For the simplex directions \mathbf{d}^x and \mathbf{d}^y found in Example 8.3, their respective reduced costs are

$$\begin{aligned}\bar{c}_x &= 13 + 0(-4) + 0(-1) + 0(-3) \\ &= 13, \\ \bar{c}_y &= 5 + 0(-1) + 0(-3) + 0(-2) \\ &= 5.\end{aligned}$$

Both are improving feasible directions.

In Example 8.5, both simplex directions were improving, so which one should be chosen? In theory, it does not matter since the algorithm will stop only when either an optimal solution or an unbounded improving direction is found. There is no hard rule for selecting an improving simplex direction because no rule has been theoretically proven to be better than another, although computational experiments suggest that some (intricate) rules typically perform better than others.

For the rest of this chapter, we will use a greedy rule for selecting our improving simplex direction. If we have a maximization problem, we choose the simplex direction whose reduced cost is most positive; if we have a minimization problem, we choose the simplex direction whose reduced cost is most negative. This rule is called the **Dantzig rule** after George Dantzig, the founder of the simplex method.

■ EXAMPLE 8.6

In Example 8.5, using Dantzig's rule, we choose simplex direction \mathbf{d}^x since its reduced cost \bar{c}_x is most positive.

In the event that there are no improving simplex directions, how can we be sure that our current solution is optimal? Let's assume that our current basic feasible solution $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ gives simplex directions $\mathbf{d}^N(1), \mathbf{d}^N(2), \dots, \mathbf{d}^N(k)$ corresponding to nonbasic variables $x_{N(1)}, \dots, x_{N(k)}$, and that none of these directions is improving. Consider a feasible direction \mathbf{d} from \mathbf{x} that satisfies $a\mathbf{d} = \mathbf{0}$, and there is at least one coordinate in \mathbf{d} corresponding to a nonbasic variable that has positive value. It can be shown that

$$\mathbf{d} = d_{N(1)}\mathbf{d}^{N(1)} + d_{N(2)}\mathbf{d}^{N(2)} + \dots + d_{N(k)}\mathbf{d}^{N(k)},$$

that is, \mathbf{d} is a (positive) linear combination of all simplex directions and the constants are the coordinates of \mathbf{d} corresponding to the nonbasic variables (see Exercise 8.26). Since each simplex direction is nonimproving, and each

coefficient in the linear combination is positive, the direction \mathbf{d} is not improving. Note that this implies an **Optimality Condition** for a linear program, since we have to check only simplex directions and not every possible feasible direction to determine whether our current solution is optimal!

If the simplex method does not identify a simplex direction that is improving, the current solution is a global optimal solution to the linear program.

Determining the Maximum Step Size

Once we've determined the direction we're going to use, we find how far to move in that direction. Recall that, given the current solution \mathbf{x} and a feasible improving direction \mathbf{d} , we move a step size $\lambda \geq 0$ to the solution $\mathbf{x} + \lambda\mathbf{d}$, so long as all of our constraints remain satisfied. To find the largest value of λ , we determine the values of λ for which our new solution remains feasible.

In the simplex method, things work out nicely. Since we've assumed that all constraints (except variable bounds) are equality constraints, these must be satisfied by every possible feasible solution. Therefore, the only possible way that our new solution $\mathbf{x} + \lambda\mathbf{d}$ could be infeasible is if one of the coordinates is negative. Thus, the maximum step size is the largest λ such that $\mathbf{x} + \lambda\mathbf{d} \geq \mathbf{0}$.

■ EXAMPLE 8.7

In Example 8.1, we see that if our current basic feasible solution is $(0, 0, 24, 24, 23)$ with basic variables s_1, s_2, s_3 , and our improving simplex direction is $\mathbf{d} = \mathbf{d}^x = (1, 0, -4, -1, -3)$, then our next solution is

$$\mathbf{x}^1 = \begin{bmatrix} 0 \\ 0 \\ 24 \\ 24 \\ 23 \end{bmatrix} + \lambda \begin{bmatrix} 1 \\ 0 \\ -4 \\ -1 \\ -3 \end{bmatrix},$$

for some $\lambda \geq 0$. The values of λ for which this solution remains feasible satisfy

$$\begin{array}{ll}
0 + \lambda \geq 0 & \lambda \geq 0 \\
24 - 4\lambda \geq 0 & \text{or} \quad \lambda \leq 6 \\
24 - \lambda \geq 0 & \lambda \leq 24 \\
23 - 3\lambda \geq 0 & \lambda \leq \frac{23}{3}.
\end{array}$$

Note that this is the same type of calculations we made in Chapter 6 (see Example 6.7). Thus, the largest value of λ that satisfies each of these inequalities is $\lambda = 6$, resulting in our new solution

$$\mathbf{x}^1 = \begin{bmatrix} 0 \\ 0 \\ 24 \\ 24 \\ 23 \end{bmatrix} + 6 \begin{bmatrix} 1 \\ 0 \\ -4 \\ -1 \\ -3 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \\ 0 \\ 18 \\ 5 \end{bmatrix}.$$

This is basic feasible solution with basis $\mathcal{B} = \{x, s_2, s_3\}$.

In Example 8.7, the one positive coordinate in \mathbf{d} played no role in determining the maximum step size. Since any inequality examined will be of the form

$$(\text{nonnegative number}) + k\lambda \geq 0,$$

the only time such an inequality plays a role in finding the maximum value of λ is if $k < 0$. This insight generates the following rule:

Ratio Test Starting at the basic feasible solution \mathbf{x} , if any coordinate of the improving simplex direction \mathbf{d}^k is negative, the maximum step size is

$$(8.6) \quad \lambda_{\max} = \min \left\{ \frac{x_j}{-d_j^k} : d_j^k < 0 \right\}.$$

■ EXAMPLE 8.8

In Example 8.7, for which the basic feasible solution was $\mathbf{x} = (0, 0, 24, 24, 23)$ and simplex direction was $\mathbf{d}^x = (1, 0, -4, -1, -3)$, the Ratio Test (8.6) generates

$$\lambda_{\max} = \min \left\{ \frac{24}{-(-4)}, \frac{24}{-(-1)}, \frac{23}{-(-3)} \right\} = 6.$$

Note that by moving in simplex direction \mathbf{d} a step size of $\lambda_{\max} > 0$, at least one of the coordinates that had positive value becomes zero, while the

nonbasic variable corresponding to \mathbf{d} will have positive value. In addition, the new solution will be another basic feasible solution (see Exercise 8.27).

What happens if there is no negative component in \mathbf{d} ?

■ EXAMPLE 8.9

Suppose that for some linear program the current basic feasible solution $(0, 0, 1, 2, 3)$ and an improving simplex direction is $\mathbf{d} = (1, 0, 2, 4, 3)$. Our next solution is

$$\mathbf{x}^1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} + \lambda \begin{bmatrix} 1 \\ 0 \\ 2 \\ 4 \\ 3 \end{bmatrix}.$$

Note that $\mathbf{x}^1 \geq \mathbf{0}$ for all $\lambda \geq 0$. Thus, we have found an unbounded direction to the feasible region that is also improving, and so our linear program is unbounded.

Test for Unbounded Linear Programs If all coordinates of an improving simplex direction \mathbf{d} are nonnegative, then the linear program is unbounded.

Updating the Basis

Once we've found the step size needed to find a subsequent basic feasible solution, we update the lists of basic and nonbasic variables, that is, we update the basis.

■ EXAMPLE 8.10

In Example 8.7, we moved from the basic feasible solution $(0, 0, 24, 24, 23)$ with basis $\mathcal{B}_0 = \{s_1, s_2, s_3\}$ to the basic feasible solution $(6, 0, 0, 18, 5)$ with basis $\mathcal{B}_1 = \{x, s_2, s_3\}$. Note that x has become basic and that our simplex direction corresponded to x . Also, the variable s_1 went from being basic with positive value to equaling zero in the new basic feasible solution, which makes it our new nonbasic variable. In addition, variable s_1 corresponds to the inequality generating the maximum step size, or equivalently, the variable

with the minimum ratio (8.6).

What we saw in Example 8.10 is typical. A (basic) variable that defines λ_{\max} becomes nonbasic. Also, a nonbasic variable will become basic because its nonnegativity constraint may become inactive.

After each simplex iteration,

- the nonbasic variable corresponding to the chosen simplex direction enters the basis and becomes basic, and
- any one of the (possibly several) basic variables that define the maximum step size will leave the basis and become nonbasic.

Entering and Leaving Variables The nonbasic variable that becomes basic is known as the *entering variable*, and the basic variable that becomes nonbasic is known as the *leaving variable*.

■ EXAMPLE 8.11

Looking at Example 8.10, variable x is the entering variable and variable s_1 is the leaving variable.

Entire Simplex Method

We have just seen the main ideas of the simplex method. It is a specialized version of the general improving search method that moves from one basic feasible solution to another until either an unbounded direction is identified or no simplex direction is improving. A formal version of the simplex method, which we call the **Basic Simplex Method** is Algorithm 8.1.

Algorithm 8.1 Basic Simplex Method

Step 0: Initialization. Identify a basic feasible solution $\mathbf{x}^{(0)}$, and set solution index $t \leftarrow 0$.

Step 1: Construct Simplex Directions. For each nonbasic variable x_j , construct the corresponding simplex direction \mathbf{d}^j using (8.4) and its reduced cost \bar{c}_j .

Step 2: Optimality Check. If no simplex direction is improving, STOP. The current solution $\mathbf{x}^{(t)}$ is optimal. Otherwise, choose any improving simplex direction as \mathbf{d} , and let x_e denote the entering variable.

Step 3: Compute Maximum Step Size. If $\mathbf{d} \geq \mathbf{0}$, stop. The linear program is unbounded. Otherwise, choose the leaving value x_l by computing the maximum step size according to the Ratio Test (8.6).

Step 4: Update Solution and Basis. Compute the new solution

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \lambda_{\max} \mathbf{d}$$

and replace x_l by x_e in the basis. Set $t \leftarrow t + 1$ and return to Step 1.

■ EXAMPLE 8.12

We continue with the problem examined in Examples 8.1–8.10. If our current basic feasible solution is $(6, 0, 0, 18, 5)$ with basis $\mathcal{B} = \{x, s_2, s_3\}$, our first step is to obtain simplex directions for the nonbasic variables y and s_1 . Using the fact that

$$B = \begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 1 & 1 \\ 3 & 0 \\ 2 & 0 \end{bmatrix}.$$

The simplex direction for variable y is of the form $\mathbf{d}^y = (d_x, 1, 0, d_{s_2}, d_{s_3})$, where $\mathbf{d}_B = (d_x, d_{s_2}, d_{s_3})$ is found by solving

$$\begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_{s_2} \\ d_{s_3} \end{bmatrix} = - \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}.$$

This yields the direction $\mathbf{d}^y = (-\frac{1}{4}, 1, 0, -\frac{11}{4}, -\frac{5}{4})$, with reduced cost $\bar{c}_y = \frac{7}{4}$, and hence it is an improving direction. For variable s_1 , the basic variable components of the simplex direction $\mathbf{d}^{s_1} = (d_x, 0, 1, d_{s_2}, d_{s_3})$ satisfy

$$\begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_{s_2} \\ d_{s_3} \end{bmatrix} = - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

This gives $\mathbf{d}^{s_1} = (-\frac{1}{4}, 1, 0, \frac{1}{4}, \frac{3}{4})$ with reduced cost $\bar{c}_{s_1} = -\frac{13}{4}$. Therefore, the only improving simplex direction is $\mathbf{d} = \mathbf{d}^y = (-\frac{1}{4}, 1, 0, -\frac{11}{4}, -\frac{5}{4})$, and our entering variable is y . By the Ratio Test (8.6), our step size is

$$\begin{aligned} \lambda_{\max} &= \min \left\{ \frac{6}{-(-\frac{1}{4})}, \frac{18}{-(-\frac{11}{4})}, \frac{5}{-(-\frac{5}{4})} \right\} \\ &= \min\{24, \frac{72}{11}, 4\} \\ &= 4. \end{aligned}$$

Our leaving variable is then s_3 . Our new basic feasible solution is

$$\begin{bmatrix} 6 \\ 0 \\ 0 \\ 18 \\ 5 \end{bmatrix} + 4 \begin{bmatrix} -\frac{1}{4} \\ 1 \\ 0 \\ -\frac{11}{4} \\ -\frac{5}{4} \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 0 \\ 7 \\ 0 \end{bmatrix}$$

with basis $\mathcal{B} = \{x, y, s_2\}$.

We continue by computing the simplex directions for nonbasic variables s_1 and s_3 . The simplex direction for s_1 is $\mathbf{d}^{s_1} = (d_x, d_y, 1, d_{s_2}, 0)$ and its basic variable components satisfy

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_{s_2} \end{bmatrix} = - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

yielding the direction $\mathbf{d}^{s_1} = (-\frac{2}{5}, \frac{3}{5}, 1, -\frac{7}{5}, 0)$ with reduced cost $\bar{c}_{s_1} = -\frac{11}{5}$. The simplex direction for s_3 is found by solving

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_{s_2} \end{bmatrix} = - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Hence, the simplex direction is $\mathbf{d}^{s_3} = (\frac{1}{5}, -\frac{4}{5}, 0, \frac{11}{5}, 1)$ with reduced cost $\bar{c}_{s_3} = -\frac{7}{5}$. Since neither simplex direction is improving, the current solution $(5, 4, 0, 7, 0)$ is optimal.

In terms of our characterization of algorithms in Chapter 5, the simplex method is a local search algorithm, where at each iteration the neighborhood of the current solution is all adjacent basic feasible solutions. In fact, even though we have presented the simplex method as a continuous optimization algorithm, we can also view it as a discrete algorithm, where we are exchanging one nonbasic variable for a basic variable at each iteration. This illustrates that the simplex method is also an exchange-type algorithm such as 2-opt and k -opt for the traveling salesperson problem.

Because in each iteration of the simplex method we exchange a variable in basis \mathcal{B} with the set of nonbasic variables \mathcal{N} , each iteration is often referred to as a **pivot**.

8.2 MAKING THE SIMPLEX METHOD MORE EFFICIENT

There is a fair amount of work done in each iteration of the simplex method. For example, in order to compute every simplex direction, we must solve $n - m$ systems of linear equations. As the number of variables (n) increases, this accounts for the majority of the time spent solving the problem. We now examine the simplex method more closely and introduce a new implementation that simplifies the amount of work needed in each iteration.

Consider the reduced cost calculations. Once we have a simplex direction \mathbf{d}^k , we compute the reduced cost

$$\bar{c}_k = c_k + \mathbf{c}_B^T \mathbf{d}_B^k,$$

where \mathbf{d}_B^k is the vector found by solving (8.4). This is because the vector \mathbf{d}^k has a 1 in coordinate k , 0 in the coordinates corresponding to other nonbasic variables, and the basic variable components are in fact the vector \mathbf{d}_B^k . Theoretically, we can also compute the reduced cost as

$$\begin{aligned}\bar{c}_k &= c_k + \mathbf{c}_B^T (-B^{-1} \mathbf{a}_k) \\ &= c_k - \mathbf{c}_B^T B^{-1} \mathbf{a}_k.\end{aligned}$$

This last formulation is interesting because it says that the vector $\mathbf{c}_B^T B^{-1}$ appears in the computation of each reduced cost \bar{c}_k . This means we do not need to know every simplex direction to compute the reduced costs! Think of the implications. Before, we had to solve $n - m$ systems of equations to find all possible simplex directions and then compute their reduced costs to see which (if any) were improving. Instead, we can compute the reduced costs first, find the entering variable, and compute only its simplex direction. We go from solving $n - m$ systems of equations to just 1! To do this, we must first find the vector $\mathbf{c}_B^T B^{-1}$, which is just the solution \mathbf{y} to the system of equations

$$(8.7) \quad \mathbf{y}^T B = \mathbf{c}_B^T \quad \text{or} \quad B^T \mathbf{y} = \mathbf{c}_B.$$

Hence, our first step is to solve (8.7) for \mathbf{y} . Then, we can compute a vector of reduced costs

$$\bar{\mathbf{c}}_N^T = \mathbf{c}_N^T - \mathbf{y}^T N.$$

The vector \mathbf{y} is a multiplier of the rows of N and is called the vector of **simplex multipliers**. We can choose our entering variable x_e and then compute its simplex direction by solving

$$B\mathbf{d}_B^e = -\mathbf{a}_e.$$

■ EXAMPLE 8.13

To construct the vector of reduced costs $\bar{\mathbf{c}}_N$ in Example 8.3 for the basic feasible solution $(0, 0, 24, 24, 23)$ with basis $\mathcal{B} = \{s_1, s_2, s_3\}$, we first solve [equation\(8.7\)](#), that is,

$$[\mathbf{y}_1 \quad \mathbf{y}_2 \quad \mathbf{y}_3] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad 0].$$

This yields the solution $\mathbf{y}^T = [0 \ 0 \ 0]$. Our reduced costs are

$$\begin{aligned} \bar{\mathbf{c}}_N^T &= [13 \quad 5] - [0 \quad 0 \quad 0] \begin{bmatrix} 4 & 1 \\ 1 & 3 \\ 3 & 2 \end{bmatrix} \\ &= [13 \quad 5], \end{aligned}$$

which are exactly the reduced costs of x and y found in Example 8.5. Using Dantzig's rule for determining the entering variable, we choose x , and its simplex direction satisfies

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_{s_1} \\ d_{s_2} \\ d_{s_3} \end{bmatrix} = - \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix},$$

which yields $\mathbf{d}^x_B = (-4, -1, -3)$.

In order to find the step size, note that the Ratio Test (8.6) only uses components of \mathbf{d}_B^k computed in (8.4). Hence, we can modify (8.6) to be

$$(8.8) \quad \lambda_{\max} = \min \left\{ \frac{(x_B)_j}{-(d_B^e)_j} : (d_B^e)_j < 0 \right\}.$$

In our new basic feasible solution, the entering variable x_e has value $x_e = \lambda_{\max}$. Once we've determined the step size and the leaving variable, we can update our basis \mathcal{B} , our nonbasic variables list \mathcal{N} , the matrices B and N , the objective vectors \mathbf{c}_B and \mathbf{c}_N , and our solution \mathbf{x}_B . Note that the vector \mathbf{x}_N of nonbasic variable values never changes, only the variables list.

■ EXAMPLE 8.14

Continuing with Example 8.13, we find that x is the entering variable and that $\mathbf{d}^x_B = (-4, -1, -3)$. Our step size is

$$\lambda_{\max} = \min \left\{ \frac{(x_B)_j}{-(d_B^e)_j} : (d_B^e)_j < 0 \right\}.$$

giving s_1 as our leaving variable. By construction, in our new solution, we have $x = 6$. Our old basic variables would have value

$$\begin{bmatrix} 24 \\ 24 \\ 23 \end{bmatrix} + 6 \begin{bmatrix} -4 \\ -1 \\ -3 \end{bmatrix} = \begin{bmatrix} 0 \\ 18 \\ 5 \end{bmatrix}.$$

Our new basis is $B = \{x, s_2, s_3\}$ and the set of nonbasic variables is $N = \{y, s_1\}$. Our new $\mathbf{x}_B = (6, 18, 5)$, the vectors \mathbf{c}_B and \mathbf{c}_N become $\mathbf{c}_B = (13, 0, 0)$, $\mathbf{c}_N = (5, 0)$, and the matrices B and N become

$$B = \begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 1 \\ 3 & 0 \\ 2 & 0 \end{bmatrix}.$$

This new implementation of the simplex method is given in Algorithm 8.2.

Algorithm 8.2 Simplex Method (Maximization Problem)

Step 0: Initialization. Find an initial basic feasible solution. Let B be the basis and N be the set of nonbasic variables. Let B (N) be the matrix whose columns correspond to the basic (nonbasic) variables, let \mathbf{c}_B (\mathbf{c}_N) be the vector of objective function coefficients associated with the basic (nonbasic) variables, and let \mathbf{x}_B (\mathbf{x}_N) denote the vector of basic (nonbasic) variable values.

Step 1: Compute Simplex Multipliers and Vector of Reduced Costs. Compute simplex multipliers \mathbf{y} by solving the system of equations

$$\mathbf{y}^T B = \mathbf{c}_B^T$$

for \mathbf{y} . Compute reduced costs

$$\bar{\mathbf{c}}_N^T = \mathbf{c}_N^T - \mathbf{y}^T N.$$

Step 2: Optimality check. If $\bar{\mathbf{c}}_N \leq \mathbf{0}$, STOP. The current solution $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ is optimal.

Step 3: Compute Simplex Direction. Otherwise, let x_e be a nonbasic variable with $\bar{c}_e > 0$. Variable x_e is our entering variable. Compute the direction \mathbf{d}_B^e by solving the system of equations

$$B \mathbf{d}_B^e = -A_e.$$

Step 4: Compute Maximum Step Size. If all coordinates of \mathbf{d} are nonnegative, STOP. The linear program is unbounded. Otherwise, choose the leaving value x_l by computing the maximum step size

according to

$$\lambda_{\max} = \min \left\{ \frac{(x_B)_j}{-(d_B^e)_j} : (d_B^e)_j < 0 \right\}.$$

Step 5: Update Solution and Basis. Update \mathcal{B} , \mathcal{N} , B , N , \mathbf{x}_B , \mathbf{x}_N , \mathbf{c}_B , \mathbf{c}_N . Return to Step 1.

■ EXAMPLE 8.15

If we continue from Example 8.14, we compute the simplex multipliers \mathbf{y} by solving

$$[y_1 \quad y_2 \quad y_3] \begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} = [13 \quad 0 \quad 0],$$

giving $\mathbf{y} = \left(\frac{13}{4}, 0, 0\right)$. Our reduced cost vector is

$$\begin{aligned} \bar{\mathbf{c}}_N^T &= [5 \quad 0] - \left[\frac{13}{4} \quad 0 \quad 0\right] \begin{bmatrix} 1 & 1 \\ 3 & 0 \\ 2 & 0 \end{bmatrix} \\ &= \left[\frac{7}{4} \quad -\frac{13}{4}\right]. \end{aligned}$$

Since $\bar{c}_y > 0$, y is our entering variable and its corresponding simplex direction \mathbf{d}_B^y satisfies

$$\begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_x \\ d_{s_2} \\ d_{s_3} \end{bmatrix} = - \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix},$$

giving $\mathbf{d}_B^y = (-\frac{1}{4}, -\frac{11}{4}, -\frac{5}{4})$. The step size is

$$\lambda_{\max} = \min \left\{ \frac{6}{\left(\frac{1}{4}\right)}, \frac{18}{\left(\frac{11}{4}\right)}, \frac{5}{\left(\frac{5}{4}\right)} \right\} = 4,$$

making s_3 our leaving variable. Updating gives $\mathcal{B} = \{x, y, s_2\}$, $\mathcal{N} = \{s_1, s_3\}$, our new $\mathbf{x}_B = (5, 4, 7)$, the vectors \mathbf{c}_B and \mathbf{c}_N become $\mathbf{c}_B = (13, 5, 0)$ and $\mathbf{c}_N = (0, 0)$, respectively, and the matrices B and N become

$$B = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

To compute our reduced costs, we first compute the simplex multipliers by solving

$$[y_1 \ y_2 \ y_3] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix} = [13 \ 5 \ 0],$$

which yields $\mathbf{y} = \left(\frac{11}{5}, 0, \frac{7}{5}\right)$. The reduced costs are

$$\begin{aligned} \bar{\mathbf{c}}_N^T &= [0 \ 0] - \left[\frac{11}{5} \ 0 \ \frac{7}{5}\right] \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \left[-\frac{11}{5} \ -\frac{7}{5}\right]. \end{aligned}$$

Since all reduced costs are negative, our current solution $\mathbf{x} = (x, y, s_1, s_2, s_3) = (5, 4, 0, 7, 0)$ is optimal.

The partition induced by the basis \mathcal{B} gives

$$B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b},$$

which can conveniently be rewritten as

$$\mathbf{x}_B = B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N.$$

By substituting this into the objective, we find

$$\begin{aligned} \mathbf{c}_B^T \mathbf{x}_B + \mathbf{c}_N^T \mathbf{x}_N &= \mathbf{c}_B^T (B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N) + \mathbf{c}_N^T \mathbf{x}_N \\ &= \mathbf{c}_B^T B^{-1}\mathbf{b} + (\mathbf{c}_N^T - \mathbf{c}_B^T B^{-1}N)\mathbf{x}_N \\ &= \mathbf{y}^T \mathbf{b} + (\mathbf{c}_N^T - \mathbf{y}^T N)\mathbf{x}_N. \end{aligned}$$

The coefficients of \mathbf{x}_N are the reduced cost vector $\bar{\mathbf{c}}_N^T$. Thus, we have all the needed information regarding our current solution by noting the basic and nonbasic variables.

8.3 CONVERGENCE, DEGENERACY, AND THE SIMPLEX METHOD

Throughout our discussion of the simplex method, we've implicitly assumed that each iteration moves from a basic feasible solution to a basic feasible solution that improves the objective function value. The algorithm eventually produces an optimal solution or shows that the linear program is unbounded as long as the objective value changes at each iteration.

If, in each iteration of the simplex method, we move from one basic feasible solution to another with an improved objective function value, then the algorithm either stops after a finite number of steps at a global optimal solution or indicates that the linear program is unbounded.

Unfortunately, it is common for a simplex direction to yield a “movement” that produces a step length of zero. How can this occur? It turns out that our old nemesis *degeneracy* is to blame. Recall that for a linear program in canonical form (8.1), a basic feasible solution is *degenerate* if there are less than m positive coordinates. This implies that at least one basic variable has value zero at the current solution. Geometrically, the solution \mathbf{x} is located at the intersection of more than n constraints and/or variable bounds. In this case, the solution \mathbf{x} is represented by multiple sets of basic variables, that is, more than one basic feasible solution corresponds to the same extreme solution.

■ EXAMPLE 8.16

From Example 7.16, the basic feasible solution $(0, 4, 0, 3, 0)$ with basis $\mathcal{B} = \{y, s_1, s_2\}$ is degenerate for

$$\begin{aligned} \max \quad & 10x + 3y \\ \text{s.t.} \quad & \\ & x + y + s_1 = 4 \\ & 5x + 2y + s_2 = 11 \\ & y + s_3 = 4 \\ & x, y, s_1, s_2, s_3 \geq 0. \end{aligned}$$

Given our basis \mathcal{B} and nonbasic variables $\mathcal{N} = \{x, s_3\}$, the reduced costs are

$$\begin{aligned} \bar{\mathbf{e}}_N^T &= [10 \ 0] - [3 \ 0 \ 0] \begin{bmatrix} 1 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 5 & 0 \\ 0 & 1 \end{bmatrix} \\ &= [10 \ -3]. \end{aligned}$$

This indicates that x is the entering variable and its simplex direction is $\mathbf{d}^x = (1, 0, -1, -5, 0)$. Our step size is

$$\lambda_{\max} = \min \left\{ \frac{0}{-(-1)}, \frac{3}{-(-5)} \right\} = \min \left\{ 0, \frac{3}{5} \right\} = 0,$$

and s_1 is the leaving variable. Our new basic feasible solution is $(0, 4, 0, 3, 0)$ with basis $\mathcal{B} = \{x, y, s_2\}$. We have not changed extreme points but did change basic feasible solutions. This pivot yielded no improvement in the objective function value.

With degeneracy it is possible for the simplex method to have nonimproving pivots. However, even though many real-world models are degenerate, this is not a great concern. Often, the simplex method will find the appropriate set of basic variables so that a “truly improving” simplex direction can be found.

Unfortunately, some extreme examples have been produced where degenerate linear programs have failed to stop. In these rare cases, the simplex method “cycles” over a set of bases that all represent the same extreme point. Of course, this occurs only if the simplex method uses the same rule each time to determine an improving simplex direction.

■ EXAMPLE 8.17

Consider the following linear program first given by Beale [11] :

$$\begin{aligned} \max \quad & 0.75x_4 - 20x_5 + 0.5x_6 - 6x_7 \\ \text{s.t.} \quad & \\ x_1 \quad & + 0.25x_4 - 8x_5 - x_6 + 9x_7 = 0 \\ x_2 \quad & + 0.5x_4 - 12x_5 - 0.5x_6 + 3x_7 = 0 \\ x_3 \quad & + x_6 = 1 \\ x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0. \end{aligned}$$

An initial basic feasible solution is $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_1, x_2, x_3\}$ and $\mathcal{N} = \{x_4, x_5, x_6, x_7\}$, which is degenerate. The reduced cost vector is

$$\bar{\mathbf{c}}_N^T = [0.75 \quad -20 \quad 0.5 \quad -6],$$

and x_4 is selected as the entering variable. Its simplex direction is $\mathbf{d}_B^4 = (-0.25, -0.5, 0)$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{0}{0.25}, \frac{0}{0.5} \right\} = 0.$$

We choose x_1 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_2, x_3, x_4\}$ and $\mathcal{N} = \{x_1, x_5, x_6, x_7\}$.

For iteration $t = 1$, the reduced cost vector is

$$\bar{\mathbf{c}}_N^T = [-3 \quad 4 \quad 3.5 \quad -33],$$

and x_5 is selected as the entering variable. Its simplex direction is $\mathbf{d}_B^5 = (-4, 32, 0)$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{0}{4} \right\} = 0.$$

We choose x_2 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_3, x_4, x_5\}$ and $\mathcal{N} = \{x_1, x_2, x_6, x_7\}$.

For iteration $t = 2$, the reduced cost vector is

$$\bar{\mathbf{c}}_N^T = [-1 \ -1 \ 2 \ -18],$$

and x_6 is selected as the entering variable. Its simplex direction is $\mathbf{d}_B^6 = (-1, -8, -0.375)$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{1}{1}, \frac{0}{8}, \frac{0}{0.375} \right\} = 0.$$

We choose x_4 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_3, x_5, x_6\}$ and $\mathcal{N} = \{x_1, x_2, x_4, x_7\}$.

For iteration $t = 3$, the reduced cost vector is

$$\bar{\mathbf{c}}_N^T = [2 \ -3 \ 0.75 \ 3],$$

and x_7 is selected as the entering variable. Its simplex direction is $\mathbf{d}_B^7 = (-10.5, -0.1875, 10.5)$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{1}{10.5}, \frac{0}{0.1875} \right\} = 0.$$

We choose x_5 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_3, x_6, x_7\}$ and $\mathcal{N} = \{x_1, x_2, x_4, x_5\}$.

For iteration $t = 4$, the reduced cost vector is

$$\bar{\mathbf{c}}_N^T = [1 \ -1 \ 1 \ -16],$$

and x_1 is selected as the entering variable. Its simplex direction is $\mathbf{d}_B^1 = (2, -2, -\frac{1}{3})$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{0}{2}, \frac{0}{-\frac{1}{3}} \right\} = 0.$$

We choose x_6 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_1, x_3, x_7\}$ and $\mathcal{N} = \{x_2, x_4, x_5, x_6\}$.

For iteration $t = 5$, the reduced cost vector is

$$\bar{c}_N^T = [2 \quad 1.75 \quad -0.5 \quad -44],$$

and x_2 is selected as the entering variable. Its simplex direction is $\mathbf{d}_B^2 = (3, 0, -\frac{1}{3})$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{0}{\left(\frac{1}{3}\right)} \right\} = 0.$$

We choose x_7 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_1, x_2, x_3\}$ and $\mathcal{N} = \{x_4, x_5, x_6, x_7\}$. Hence, we've cycled back to the initial basic feasible solution.

In practice, fortunately such examples are very, very rare, and it is usually safe to assume practical problems will not cycle. However, a few rules have been proposed to eliminate cycling. The simplest and easiest rule to implement was developed by Bland in 1977 [17].

Bland's Rule to Prevent Cycling:

1. Select the entering variable by choosing the nonbasic variable with the smallest index among those whose corresponding simplex direction is improving.
2. Select the leaving variable by choosing from among those the basic variable with the smallest index that defines λ_{\max} .

Bland showed that these prevent cycling and the augmented simplex method stops after a finite number of iterations.

■ EXAMPLE 8.18

Recall Example 8.17, and consider the basic feasible solution at the beginning of iteration $t = 3$, where the basis was $\mathcal{B} = \{x_3, x_5, x_6\}$. If we use Bland's rule to choose the entering variable, x_1 now enters since its reduced cost $\bar{c}_1 = 2$. Its simplex direction is $\mathbf{d}_B^1 = (-1.5, -0.0625, 1) = (-1.5, -0.0625, 1.5)$ and the step size is

$$\lambda_{\max} = \min \left\{ \frac{1}{1.5}, \frac{0}{0.0625} \right\} = 0.$$

We choose x_5 as our leaving variable, and the basic feasible solution becomes $(0, 0, 1, 0, 0, 0, 0)$ with basis $\mathcal{B} = \{x_1, x_3, x_6\}$ and $\mathcal{N} = \{x_2, x_4, x_5, x_7\}$.

For the next iteration, the reduced cost vector is

$$\bar{e}_N^T = [1 \ 1.25 \ -32 \ -3],$$

and x_2 is selected as the entering variable using Bland's rule. Its simplex direction is $d_B^2 = (2, -2, 2) = (2, -2, 2)$ and its step size is

$$\lambda_{\max} = \min \left\{ \frac{1}{(2)} \right\} = \frac{1}{2}.$$

We choose x_3 as our leaving variable, and the basic feasible solution becomes $(1, 0.5, 0, 0, 0, 1, 0)$ with basis $\mathcal{B} = \{x_1, x_2, x_6\}$ and $\mathcal{N} = \{x_3, x_4, x_5, x_7\}$.

Since we are at a new extreme point, cycling did not occur.

You may be wondering, “Is cycling the only way the simplex method does not terminate?” The answer is yes.

If the simplex method fails to terminate, the algorithm must be cycling through a sequence of basic feasible solutions all corresponding to the same extreme point.

8.4 FINDING AN INITIAL SOLUTION: TWO-PHASE METHOD

In Section 8.1, we assumed an initial basic feasible solution was available. This is especially true if we start with a linear program in standard form

$$\begin{aligned} & \max e^T x \\ & \text{s.t.} \\ & Ax \leq b \\ & x \geq 0, \end{aligned}$$

for which $b \geq 0$, where an initial basic feasible solution for its canonical form is the vector $(0, b)$ with basis $\mathcal{B} = \{s_1, s_2, \dots, s_m\}$.

When an initial basic feasible solution is not apparent, what can we do? One not-so-good option is to search for a basic feasible solution by enumerating all combinations of m columns of A . Unfortunately, even if

there are only 15 constraints and 30 variables, a small problem by today's standards, there are over 155 million combinations to check.

Our problem arises when the origin (in the original variables) is not feasible. One idea for this problem is to solve an optimization problem that minimizes the infeasibility of a solution. In other words, in order to find a feasible solution to an optimization problem, solve a related optimization problem. But how do we do this? We design an “artificial problem” that measures how far our current solution is from satisfying the constraints, and which allows an easy-to-find initial solution (like the origin).

■ EXAMPLE 8.19

The problem we are considering is

$$\begin{aligned} \max \quad & 6x + 5y \\ \text{s.t.} \quad & -5x + y \leq 2 \\ & 2x + 5y \leq 44 \\ & 3x + y \leq 27 \\ & 2x - 5y \leq 1 \\ & x + y \geq 4 \\ & x, y \geq 0. \end{aligned}$$

Note that the origin is not feasible. Since the right-hand side is nonnegative, the only constraint the origin does not satisfy is $x + y \geq 4$. In canonical form, our problem becomes

$$\begin{aligned} \max \quad & 6x + 5y \\ \text{s.t.} \quad & -5x + y + s_1 = 2 \\ & 2x + 5y + s_2 = 44 \\ & 3x + y + s_3 = 27 \\ & 2x - 5y + s_4 = 1 \\ & x + y - s_5 = 4 \\ (8.9) \quad & x, y, s_1, s_2, s_3, s_4, s_5 \geq 0. \end{aligned}$$

We almost have the identity matrix with the additional variables. Since the only column of the identity matrix that is missing corresponds to the constraint that the origin does not satisfy, why not add an **artificial variable**, a_5 in this constraint in order to give us the identity matrix, and hence a basic

feasible solution to a new linear program? In this model, the infeasibility is measured by , a_5 ; if , $a_5 > 0$, then our current solution is not feasible to (8.9), but if , $a_5 \leq 0$ it is. The artificial linear program is

$$\begin{aligned} \min \quad & a_5 \\ \text{s.t.} \quad & -5x + y + s_1 = 2 \\ & 2x + 5y + s_2 = 44 \\ & 3x + y + s_3 = 27 \\ & 2x - 5y + s_4 = 1 \\ & x + y - s_5 + a_5 = 4 \\ (8.10) \quad & x, y, s_1, s_2, s_3, s_4, s_5, a_5 \geq 0. \end{aligned}$$

Since (8.10) is bounded below by , $a_5 = 0$, if we achieve this value we have an optimal solution, and an initial basic feasible solution can be obtained by deleting the column corresponding to the artificial variable , a_5 .

This approach to finding a feasible solution to a linear program is part of what is known as the **Two-Phase Method**. In Phase I, an artificial linear program is formulated and solved to find a basic feasible solution to the original problem. In Phase II, the original problem is solved to optimality starting with the initial basic feasible solution obtained in Phase I. Both problems can be solved using the simplex method.

■ EXAMPLE 8.20

Let's solve the linear program in (8.10). Our initial basic feasible solution is

$$\mathbf{x}^0 = \begin{bmatrix} x \\ y \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ a_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 44 \\ 27 \\ 1 \\ 0 \\ 4 \end{bmatrix}$$

with basis $\mathcal{B} = \{s_1, s_2, s_3, s_4, a_5\}$. Computing the reduced cost vector gives

$$\bar{\mathbf{c}}_N^T = [-1 \quad -1 \quad 1].$$

We let x be our entering variable (recall that we're minimizing, so we pick a

negative reduced cost). The corresponding simplex direction is then $\mathbf{d}_B^x = (5, -2, -3, -2, -1)$. The Ratio Test gives

$$\lambda_{\max} = \min \left\{ \frac{44}{2}, \frac{27}{3}, \frac{1}{2}, \frac{4}{1} \right\} = \frac{1}{2},$$

with s_4 as our leaving variable, giving the solution $\mathbf{x}^1 = (\frac{1}{2}, 0, \frac{9}{2}, 43, \frac{51}{2}, 0, 0, \frac{7}{2})$ with basis $\mathcal{B} = \{x, s_1, s_2, s_3, a_5\}$. Computing the reduced cost vector gives

$$\bar{\mathbf{c}}_N^T = \begin{bmatrix} -\frac{7}{2} & \frac{1}{2} & 1 \end{bmatrix},$$

giving y as our entering variable. Its simplex direction is $\mathbf{d}_B^y = (\frac{5}{2}, \frac{23}{2}, -10, -\frac{17}{2}, -\frac{7}{2})$. Our leaving variable is a_5 , since it corresponds to the value of λ_{\max} found from the Ratio Test:

$$\lambda_{\max} = \min \left\{ \frac{43}{10}, \frac{\left(\frac{51}{2}\right)}{\left(\frac{17}{2}\right)}, \frac{\left(\frac{7}{2}\right)}{\left(\frac{7}{2}\right)} \right\} = 1.$$

Our new solution then is $\mathbf{x}^2 = (3, 1, 16, 33, 17, 0, 0, 0)$. Since $a_5 = 0$, we have solved the Phase I linear program, and a basic feasible solution to (8.9) is $(3, 1, 16, 33, 17, 0, 0)$ with basis $\mathcal{B} = \{x, y, s_1, s_2, s_3\}$. We now remove a_5 from the constraints, restore the original objective function, and begin the Phase II problem from initial basic feasible solution $\mathbf{x} = (3, 1, 16, 33, 17, 0, 0)$.

The key to the Phase I problem is that the solution where the original (nonslack) variables are equal to 0 is feasible. We can easily formulate this problem by following some basic rules.

To formulate the Phase I linear program

1. Rewrite your linear program into a canonical form with the right-hand side $\mathbf{b} \geq \mathbf{0}$.
2. If there is no slack variable (with coefficient +1) in row i , add an artificial variable a_i in row i . If there is a slack variable in row i , do not add any artificial variable.
3. The objective function is to minimize the sum of the added artificial variables.
4. The initial basic feasible solution for Phase I can be found by placing

all the slack and artificial variables into the basis.

■ EXAMPLE 8.21

Consider

$$\begin{aligned} \max \quad & 6x + 5y \\ \text{s.t.} \quad & -5x + y \leq -2 \\ & 2x + 5y \leq 44 \\ & 3x - y = -1 \\ & 2x - 5y \leq 27 \\ & x + y \geq 4 \\ & x, y \geq 0. \end{aligned}$$

Its canonical form, with nonnegative right-hand side, is

$$\begin{aligned} \max \quad & 6x + 5y \\ \text{s.t.} \quad & 5x - y - s_1 = 2 \\ & 2x + 5y + s_2 = 44 \\ & -3x + y = 1 \\ & 2x - 5y + s_4 = 27 \\ & x + y - s_5 = 4 \\ & x, y, s_1, s_2, s_4, s_5 \geq 0. \end{aligned}$$

To formulate the Phase I problem, we'd add artificial variables , a_1 to the first constraint, , a_3 to the third constraint, and , a_5 to the fifth constraint, giving the linear program

$$\begin{aligned} \min \quad & a_1 + a_3 + a_5 \\ \text{s.t.} \quad & 5x - y - s_1 + a_1 = 2 \\ & 2x + 5y + s_2 = 44 \\ & -3x + y + a_3 = 1 \\ & 2x - 5y + s_4 = 27 \\ & x + y - s_5 + a_5 = 4 \\ & x, y, s_1, s_2, s_4, s_5, a_1, a_3, a_5 \geq 0. \end{aligned}$$

Our initial basic feasible solution to the Phase I problem is

$$(x, y, s_1, s_2, s_4, s_5, a_1, a_3, a_5) = (0, 0, 0, 44, 27, 0, 2, 1, 4).$$

Returning to Example 8.20, what if our Phase I optimal value had been positive?

Rule If the optimal solution to the Phase I problem is positive, then the original linear program has no feasible solutions.

■ EXAMPLE 8.22

Consider the following linear program:

$$\max \quad 13x + 5y$$

s.t.

$$3x + 2y \leq 12$$

$$x + y \leq 5$$

$$x \geq 3$$

$$y \geq 2$$

$$x, y \geq 0.$$

This linear program has no feasible solution. Its Phase I linear program is

$$\min \quad a_3 + a_4$$

s.t.

$$3x + 2y + s_1 = 12$$

$$x + y + s_2 = 5$$

$$x - s_3 + a_3 = 3$$

$$y - s_4 + a_4 = 2$$

$$x, y, s_1, s_2, s_3, s_4, a_3, a_4 \geq 0,$$

and the initial basic feasible solution is $(0, 0, 12, 5, 0, 0, 3, 2)$ with basis $\mathcal{B} = \{s_1, s_2, a_3, a_4\}$. The reduced cost vector is

$$\bar{c}^T = [-1 \quad -1 \quad 1 \quad 1],$$

giving x as our entering variable. Its simplex direction is $d_B^x = (-3, -1, -1, 0)$. The Ratio Test yields

$$\lambda_{\max} = \min \left\{ \frac{12}{3}, 5, 3 \right\} = 3,$$

giving us the leaving variable a_3 and a new solution $(3, 0, 3, 2, 0, 0, 0, 2)$ with basis $\mathcal{B} = \{x, s_1, s_2, a_4\}$. The reduced cost vector is

$$\bar{c}_N^T = [-1 \quad 0 \quad 1 \quad 1],$$

giving y as our entering variable. Its simplex direction is $\mathbf{d}_B^y = (0, -2, -1, -1) = (0, -2, -1, -1)$. The Ratio Test yields

$$\lambda_{\max} = \min \left\{ \frac{3}{2}, 2, 2 \right\} = \frac{3}{2},$$

giving us the leaving variable s_1 and a new solution $(3, \frac{3}{2}, 0, \frac{1}{2}, 0, 0, 0, \frac{1}{2})$ with basis $\mathcal{B} = \{x, s_1, s_2, a_4\}$. The reduced cost vector is

$$\bar{\mathbf{c}}_N^T = \left[\frac{1}{2} \quad \frac{3}{2} \quad 1 \quad -\frac{1}{2} \right],$$

giving a_3 as our entering variable. Its simplex direction is $\mathbf{d}_B^{a_3} = (-1, \frac{3}{2}, -\frac{1}{2}, -\frac{3}{2})$. The Ratio Test yields

$$\lambda_{\max} = \min \left\{ 3, \frac{\left(\frac{3}{2}\right)}{\left(\frac{3}{2}\right)}, \frac{\left(\frac{1}{2}\right)}{\left(\frac{1}{2}\right)}, \frac{\left(\frac{1}{2}\right)}{\left(\frac{3}{2}\right)} \right\} = \frac{1}{3},$$

giving us the leaving variable a_4 and a new solution $(\frac{8}{3}, 2, 0, \frac{1}{3}, 0, 0, \frac{1}{3}, 0)$ with basis $\mathcal{B} = \{x, s_1, s_2, a_3\}$. The reduced cost vector is

$$\bar{\mathbf{c}}_N^T = \left[\frac{1}{3} \quad 1 \quad \frac{2}{3} \quad \frac{1}{3} \right],$$

indicating that we have found an optimal solution. However, because $a_3 > 0$, we know that there are no feasible solutions to the original linear program. It is possible for an artificial variable a_j to be in the basis even if the optimal value to the Phase I problem is 0. In Exercise 8.13, we examine how to handle such a case.

8.5 BOUNDED SIMPLEX METHOD

When additional conditions are placed upon a linear program, we want to reformulate the simplex method in order to improve its efficiency for this case. This is the approach we take for linear programs with variable upper bounds.

Consider the following linear program:

$$\begin{aligned}
 \max \quad & \mathbf{c}^T \mathbf{x} \\
 \text{s.t.} \quad & \\
 & A\mathbf{x} = \mathbf{b} \\
 (8.11) \quad & \mathbf{0} \leq \mathbf{x} \leq \mathbf{u},
 \end{aligned}$$

where each component u_j is positive. This linear program differs from the canonical form because we have removed any existing constraints $x_j \leq u_j$ from the constraint matrix; we shall say that such a linear program is in **bounded canonical form**. In this section, we design an improved version of the simplex method, known as the **Bounded Simplex Method**, that handles these upper bounds explicitly without adding them to the constraint matrix.

First, why should we consider doing this? If we add the upper bounds to our constraint matrix, we add more rows (and columns) to our basis matrix B , which increases the time needed to solve the resulting system of equations. Thus, we improve the efficiency of the algorithm by directly dealing with these constraints.

To construct a specialized version of the simplex method, we need to identify the characteristics of a basic feasible solution. Recall that for any polyhedral set, a solution \mathbf{x} is a basic feasible solution if it is active at n (or more) linearly independent constraints. If we are given a linear program in canonical form, \mathbf{x} is a basic feasible solution if (a) \mathbf{x} satisfies all the equality constraints and (b) the columns of the constraint matrix corresponding to positive values of x are linearly independent. Alternatively, the variables can be partitioned into $n - m$ nonbasic variables, whose values are 0, and m variables whose values are uniquely determined by the equality constraints. For a linear program in bounded canonical form, we seek an equivalent definition for a basic feasible solution.

Extended Basic Solution A solution \mathbf{x} is an *extended basic solution* for (8.11) if \mathbf{x} satisfies $A\mathbf{x} = \mathbf{b}$ and the columns of A corresponding to the components of \mathbf{x} that are strictly between their bounds are linearly independent.

Extended Basic Feasible Solution An extended basic solution \mathbf{x} is an *extended basic feasible solution* if it also satisfies the variable bounds $\mathbf{0} \leq \mathbf{x} \leq \mathbf{u}$.

Since we are assuming that the rank of A is m , we can partition the variables

of a extended basic feasible solution \mathbf{x} into $n - m$ nonbasic variables, of which k are at their lower bound of 0 and $n - m - k$ are at their upper bound, and m basic variables whose values are determined by fixing the nonbasic variables to their appropriate bound and solving the constraint equation

$$B\mathbf{x}_B = \mathbf{b} - N\mathbf{x}_N.$$

We shall denote this new basis structure by $(\mathcal{B}, \mathcal{L}, \mathcal{U})$, where \mathcal{B} denotes the set of basic variables, \mathcal{L} the set of nonbasic variables whose values are at the lower bound of 0, and \mathcal{U} the set of nonbasic variables whose values are at their upper bound.

■ EXAMPLE 8.23

Consider the linear program in bounded canonical form

$$\begin{aligned} \max \quad & 2x + 3y \\ \text{s.t.} \quad & 3x + y + s_1 = 14 \\ & -x + y + s_2 = 4 \\ & 0 \leq x \leq 4 \\ & 0 \leq y \leq 5 \\ & 0 \leq s_1 \leq 14 \\ (8.12) \quad & 0 \leq s_2 \leq 8. \end{aligned}$$

The solution $(1, 5, 6, 0)$ is an extended basic feasible solution with basic variables $\mathcal{B} = \{x, s_1\}$ and nonbasic variables $\mathcal{L} = \{s_2\}$ and $\mathcal{U} = \{y\}$. The solution $(0, 4, 10, 0)$ is an extended basic feasible solution with variable partition $\mathcal{B} = \{y, s_1\}$, $\mathcal{L} = \{x, s_2\}$, and $\mathcal{U} = \emptyset$. The extended basic feasible solution \mathbf{x} with variable partition $(\mathcal{B}, \mathcal{L}, \mathcal{U}) = (\{y, s_2\}, \{s_1\}, \{x\})$ corresponds to the solution $(4, 2, 0, 6)$ because $s_1 = 0$, $x = 4$, and y and s_2 are found by solving

$$\begin{aligned} 3(4) + y &= 14 \\ -4 + y + s_2 &= 4. \end{aligned}$$

Is this definition of an extended basic feasible solution equivalent to the one given in Chapter 7? The following states that it is the case.

Proposition 8.1 *A solution \mathbf{x} to a linear program in bounded canonical form (8.11) is an extended basic feasible solution if and only if (\mathbf{x}, \mathbf{s}) is a basic feasible solution to the linear program*

$$\begin{aligned}
& \max && \mathbf{c}^T \mathbf{x} \\
& \text{s.t.} && \\
& && A\mathbf{x} = \mathbf{b} \\
& && \mathbf{x} + \mathbf{s} = \mathbf{u} \\
& && \mathbf{x}, \mathbf{s} \geq \mathbf{0}.
\end{aligned}$$

■ EXAMPLE 8.24

For the linear program in Example 8.23, we saw that the solution $(1, 5, 6, 0)$ is an extended basic feasible solution. This corresponds to the basic feasible solution $(1, 5, 6, 0, 3, 0)$ of the linear program

$$\begin{aligned}
& \max && 2x + 3y \\
& \text{s.t.} && \\
& && 3x + y + s_1 = 14 \\
& && -x + y + s_2 = 4 \\
& && x + s_3 = 4 \\
& && y + s_4 = 5 \\
& && x, y, s_1, s_2, s_3, s_4 \geq 0.
\end{aligned}$$

The simplex method extends to this new basis structure. The first step in each iteration is to identify either an entering variable (a nonbasic variable that will become basic) or prove that the current solution is optimal; this is done by examining the reduced costs of the nonbasic variables. We saw that, for a maximization problem, if there was a Nonbasic Variable With positive reduced cost, then the corresponding simplex direction was improving, and hence we could find another feasible solution with an improved objective value. Another way to look at this is to say that if a nonbasic variable has positive reduced cost we can find a better solution if we can increase its value and remain feasible; similarly, if a nonbasic variable has negative reduced cost, if we can reduce this variable's value and remain feasible, we can increase the value of the objective function. It is this interpretation that is now useful. If we have a linear program in bounded canonical form, and we have an extended basic feasible solution with nonbasic variable x_j , $j \in \mathcal{U}$, with negative reduced cost, its corresponding simplex direction is improving! This gives the following updated optimality conditions.

Given an extended basic feasible solution \mathbf{x} with basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ to a linear program in bounded canonical form, \mathbf{x} is the optimal solution if and only if

$$1. \bar{c}_j \leq 0, j \in \mathcal{L},$$

$$2. \bar{c}_j \geq 0, j \in \mathcal{U},$$

Where

$$\bar{c}_j = c_j - \mathbf{c}_B^T B^{-1} \mathbf{a}_j$$

is the reduced cost of variable x_j .

■ EXAMPLE 8.25

From Example 8.23, we consider the extended basic feasible solution $(4, 2, 0, 6)$, where $(\mathcal{B}, \mathcal{L}, \mathcal{U}) = (\{y, s_2\}, \{s_1\}, \{x\})$. Since

$$B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad N = \begin{bmatrix} 3 & 1 \\ -1 & 0 \end{bmatrix},$$

the reduced cost vector is

$$\begin{aligned} \bar{\mathbf{c}}_N^T &= [\bar{c}_x \ \bar{c}_{s_1}] = [2 \ 0] - [3 \ 0] \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 3 & 1 \\ -1 & 0 \end{bmatrix} \\ &= [-7 \ -3]. \end{aligned}$$

Since $\bar{c}_x < 0$ and $x \in \mathcal{U}$, the solution $(4, 2, 0, 6)$ is not an optimal solution. The extended basic feasible solution $(3, 5, 0, 2)$, with $(\mathcal{B}, \mathcal{L}, \mathcal{U}) = (\{x, s_2\}, \{s_1\}, \{y\})$, has reduced costs

$$\begin{aligned} \bar{\mathbf{c}}_N^T &= [\bar{c}_y \ \bar{c}_{s_1}] = [3 \ 0] - [2 \ 0] \begin{bmatrix} 3 & 0 \\ -1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \\ &= [\frac{7}{3} \ -\frac{2}{3}]. \end{aligned}$$

Since $\bar{c}_y > 0$ ($y \in \mathcal{U}$) and $\bar{c}_{s_1} < 0$ ($s_1 \in \mathcal{L}$), this solution is optimal.

To identify the simplex direction and step size, we see that if our entering variable $x_j \in \mathcal{L}$ then computing the simplex direction is identical to our previous computation. If $x_j \in \mathcal{U}$, we decrease its value, and hence the corresponding simplex direction component is -1 . If we assume that the direction components of the other nonbasic variables are 0 , we see that the basic variable components is found by solving the system of equations

$$B\mathbf{d} - \mathbf{a}_j = \mathbf{0} \quad \text{or} \quad B\mathbf{d} = \mathbf{a}_j.$$

We need to be careful in our computation of the step size. Previously, we

needed to consider only the nonnegativity bound of each basic variable—this gave us our Ratio Test. However, we now have to consider the upper bounds of each basic variable and the lower and upper bounds of the entering variable. If \mathbf{d}^e is the basic variable component of the simplex direction and \mathbf{x}_B is the vector of basic variable values at the current solution, we must have for each basic variable x_k

$$0 \leq (x_B)_k + \lambda d_k^e \leq u_k.$$

If $d_k^e < 0$, then we solve the lower bound inequality for λ to get

$$\lambda \leq \frac{(x_B)_k}{-d_k^e}.$$

If $d_k^e > 0$, then we use the upper bound inequality to get

$$\lambda \leq \frac{u_k - (x_B)_k}{d_k^e}.$$

In addition, we have to make sure that our entering variable x_j satisfies both upper and lower bounds; since the most x_e can change (in either direction) is u_e , we also have

$$\lambda \leq u_e.$$

Thus, we get the following updated Ratio Test rule:

$$\lambda_{\max} = \min \left\{ u_e, \min \left\{ \frac{(x_B)_k}{-d_k^e} : d_k^e < 0 \right\}, \min \left\{ \frac{u_k - (x_B)_k}{d_k^e} : d_k^e > 0 \right\} \right\}.$$

Using the simplex direction and the maximum step size, a new extended basic feasible solution is identified. But again, we must be careful. If the maximum step size is $\min \left\{ \frac{(x_B)_k}{-d_k^e} : d_k^e < 0 \right\}$, then the leaving variable has a new value of 0 and becomes an element of \mathcal{L} . If the step size is $\min \left\{ \frac{u_k - (x_B)_k}{d_k^e} : d_k^e > 0 \right\}$, then the leaving variable will have a value of u_k and becomes an element of \mathcal{U} . Finally, if $\lambda_{\max} = u_e$, our entering variable will switch bounds. In this case, the basis \mathcal{B} is unchanged and entering variable x_e remains nonbasic. However, we do move to a new extended basic feasible solution.

■ EXAMPLE 8.26

In Example 8.23, we suppose that the current extended basic feasible solution

is $(4, 2, 0, 6)$ with basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U}) = (\{y, s_2\}, \{s_1\}, \{x\})$. We have already seen in Example 8.25 that we can choose $x \in \mathcal{U}$ as an entering variable. Since $x \in \mathcal{U}$, we know that the simplex direction component of x must have value -1 and the basic variable components of the direction can be calculated by solving the system of equations

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} d_y \\ d_{s_2} \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \end{bmatrix},$$

giving $\mathbf{d} = (d_y, d_{s_2}) = (3, -4)$. To compute the maximum step size, we set

$$\lambda_{\max} = \min \left\{ 4, \frac{6}{-(-4)}, \frac{5-2}{3} \right\} = 1.$$

Since λ_{\max} is defined by variable y 's calculation, y becomes nonbasic and becomes an element of \mathcal{U} . Variable x moves from \mathcal{U} to \mathcal{B} and our new solution is

$$\begin{bmatrix} 4 \\ 2 \\ 0 \\ 6 \end{bmatrix} + 1 \begin{bmatrix} -1 \\ 3 \\ 0 \\ -4 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \\ 0 \\ 2 \end{bmatrix}.$$

Now consider the extended basic feasible solution $(0, 0, 14, 4)$ with basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U}) = (\{s_1, s_2\}, \{x, y\}, \emptyset)$. The reduced cost vector is

$$\bar{\mathbf{c}}_N^T = [2 \ 3],$$

indicating that the simplex direction for x is improving. The basic variable component, of this direction satisfies

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_{s_1} \\ d_{s_2} \end{bmatrix} = - \begin{bmatrix} 3 \\ -1 \end{bmatrix}.$$

The maximum step size is

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d_{s_1} \\ d_{s_2} \end{bmatrix} = - \begin{bmatrix} 3 \\ -1 \end{bmatrix}.$$

This implies that the entering variable x remains nonbasic but leaves \mathcal{L} and becomes an element of \mathcal{U} . The new extended basic feasible solution is

$$\begin{bmatrix} 0 \\ 0 \\ 14 \\ 4 \end{bmatrix} + 4 \begin{bmatrix} 1 \\ 0 \\ -3 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 2 \\ 8 \end{bmatrix},$$

with basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U}) = (\{s_1, s_2\}, \{y\}, \{x\})$. Note that $s_2 \in \mathcal{B}$ even though its value equals its upper bound because we need to have exactly $m = 2$ basic variables.

If we put these enhancements together, we get the bounded simplex algorithm given in Algorithm 8.3.

8.6 COMPUTATIONAL ISSUES

Let's examine how various computer implementations of the simplex method work. When mathematical algorithms are implemented in computer programs, many alterations are made. These generally are due to creating more efficient implementations, but they also deal with such numerical issues as round-off errors and linear algebra concerns. In any computer implementation of an algorithm, one theme that often occurs is "efficiency." This could involve doing many iterations of an algorithm, each involving very few computational operations, or doing fewer iterations, with each requiring many operations. It is usually better to implement an algorithm that involves more operations per iteration but results in fewer iterations; this typically results in faster computational times and uses less memory. In this section, we'll explore some of the efforts that have been made to implement the simplex method as efficiently as possible.

Algorithm 8.3 Bounded Simplex Method (Maximization Problem)

Step 0: Initialization. Find an initial extended basic feasible solution \mathbf{x} . Let $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ denote the basis structure of \mathbf{x} . Let \mathbf{B} (\mathbf{N}) be the matrix whose columns correspond to the basic (nonbasic) variables, let \mathbf{c}_B (\mathbf{c}_N) be the vector of objective function coefficients associated with the basic (nonbasic) variables, and let \mathbf{x}_B (\mathbf{x}_N) denote the vector of basic (nonbasic) variable values.

Step 1: Compute Simplex Multipliers and Vector of Reduced Costs. Compute the simplex multipliers \mathbf{y} by solving

$$\mathbf{y}^T \mathbf{B} = \mathbf{c}_B^T$$

for \mathbf{y} . Compute reduced costs

$$\bar{\mathbf{c}}^T = \mathbf{c}_N^T - \mathbf{y}^T \mathbf{N}.$$

Step 2: Optimality check. If $\bar{c}_j \leq 0$ for all $j \in \mathcal{L}$ and $\bar{c}_j \geq 0$ for all $j \in \mathcal{U}$, stop. The current solution $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ is optimal.

Step 3: Compute Simplex Direction. Otherwise, let x_e be a nonbasic variable violating the above

optimality condition. Variable x_e is our entering variable. If $x_e \in \mathcal{L}$, compute the direction \mathbf{d}^e by solving the system of equations

$$B\mathbf{d}^e = -\mathbf{A}_e.$$

If $x_e \in \mathcal{U}$, compute the direction \mathbf{d} by solving the system of equations

$$B\mathbf{d}^e = \mathbf{A}_e.$$

Step 4: Compute Maximum Step Size. Computing the maximum step size according to

$$\lambda_{\max} = \min \left\{ u_e, \min \left\{ \frac{(x_B)_k}{-d_k^e} : d_k^e < 0 \right\}, \min \left\{ \frac{u_k - (x_B)_k}{d_k^e} : d_k^e > 0 \right\} \right\}.$$

Step 5: Update Solution and Basis. Update \mathcal{B} , \mathcal{L} , \mathcal{U} , B , N , \mathbf{x}_B , \mathbf{x}_N , \mathbf{c}_B , \mathbf{c}_N . Return to Step 1.

Pricing Methods

Choosing the entering variable, also known as **pricing**, is one of the more computationally expensive parts of the simplex method. Various implementations try to reduce the number of operations required to do this step, or at least be “smarter” when doing many operations. In Section 8.1, we used Dantzig’s rule for choosing the entering variable. In practice, this method is rarely, if ever, used; it is computationally too expensive since each variable’s reduced cost must be computed at each iteration, and it generally requires many iterations because directions that have large reduced cost typically have smaller step sizes, resulting in smaller net improvement per iteration. A slightly better implementation chooses the first simplex direction that is found improving.

An approach that is often implemented is that of **Partial Pricing**. In this approach, only a subset of reduced costs are initially calculated; if one of these indicates that the current solution is not optimal, that variable is chosen as the entering variable. If no improving direction is found from this subset, another subset of variables is generated and their reduced costs are calculated. This continues until either an entering variable is found or optimality is proven. While every reduced cost may be calculated in each iteration, very few iterations will calculate all of them.

A more common approach to choosing the entering variable is to explore the geometry. For maximization problems, the gradient is the improving direction that has the largest rate of improvement among all other improving directions. Since we are only considering simplex directions, we would want

the direction that makes the smallest angle with the gradient as possible. This **Steepest Edge Pricing** selects the entering variable by solving

$$\min_j \frac{\mathbf{c}^T \mathbf{d}_j}{\|\mathbf{d}_j\|},$$

over the simplex directions \mathbf{d}_j . Computational experiments have shown that choosing the entering variables using steepest edge pricing typically reduces the number of iterations (and time) the simplex method requires, but its computational cost is huge because of the need to calculate the norm $\|\mathbf{d}_j\|$ of each direction. A related approach, known as **Devex Pricing**, approximates the norm calculations in the steepest edge rule. Devex is typically implemented in many top-of-the-line linear programming software packages. For more information, see Refs. [38, 47, 56].

Finding an Initial Basis

In Section 8.4, we developed a Phase I linear program with an easy-to-identify basic feasible solution whose optimal solution identifies a basic feasible solution to our linear program. However, this approach has drawbacks. First, we need to add variables in order to start with a feasible solution. Second, we start with a basis consisting of slack and artificial variables, which will likely have no similarity to the set of basic variables in the optimal solution. To rectify this, modern computer implementations will try to identify an initial basis that uses very few artificial and slack variables. This heuristic procedure, often referred to as a “crash,” was first described by Bixby [15] and attempts to identify basic variables that are (a) as close to feasible as possible, (b) the resulting basis matrix B is as close to being triangular as possible, and (c) the variables are “likely” to be in the optimal basis. Typically, various iterations are done in these procedures.

Summary

We have now assembled the building blocks from Chapters 5, 6, and 7 and developed an algorithm for solving linear programs. In particular, the simplex method is a specific instance of a general improving search method that takes advantage of properties of linear programs. We developed a more efficient version that exploited the linear algebra of the original approach, and we

determined a two-phase approach to find an initial basic feasible solution. We then extended this method to linear programs where the variables had upper and lower bounds.

We are not done with linear programs. As operations researchers, we should look deeper into the problem for some theoretical implications. What we will find in the next few chapters is that these theoretical tools will help us do many practical things, including sensitivity analysis and the development of additional algorithms for solving linear programs.

EXERCISES

Solve the linear programs in Exercises 8.1–8.6 using the simplex method found in Algorithm 8.1. Be sure to indicate, for each simplex direction, the system of equations being solved. Use Dantzig’s rule to choose your entering variables, and make your initial basic feasible solution correspond to the nonslack variables having value 0.

8.1

$$\max \quad 8x_1 + 9x_2 + 5x_3$$

s.t.

$$x_1 + x_2 + 2x_3 \leq 2$$

$$2x_1 + 3x_2 + 4x_3 \leq 3$$

$$6x_1 + 6x_2 + 2x_3 \leq 8$$

$$x_1, x_2, x_3 \geq 0.$$

8.2

$$\max \quad 3x + 2y$$

s.t.

$$2x - y \leq 6$$

$$2x + y \leq 10$$

$$x, y \geq 0.$$

8.3

$$\begin{aligned}
& \max && x + y \\
& \text{s.t.} && \\
& && -2x + y \leq 0 \\
& && x - 2y \leq 0 \\
& && x + y \leq 9 \\
& && x, y \geq 0.
\end{aligned}$$

8.4

$$\begin{aligned}
& \max && 4x_1 + 2x_2 + 7x_3 \\
& \text{s.t.} && \\
& && 2x_1 - x_2 + 4x_3 \leq 18 \\
& && 4x_1 + 2x_2 + 5x_3 \leq 10 \\
& && x_1, x_2, x_3 \geq 0.
\end{aligned}$$

8.5

$$\begin{aligned}
& \min && 2x_1 - x_2 + 3x_3 \\
& \text{s.t.} && \\
& && x_1 - x_2 + 4x_3 \leq 8 \\
& && 2x_1 + 2x_2 - 5x_3 \leq 4 \\
& && x_1 + x_3 \leq 6 \\
& && x_1, x_2, x_3 \geq 0.
\end{aligned}$$

8.6

$$\begin{aligned}
& \max && x_1 + 2x_2 \\
& \text{s.t.} && \\
& && 4x_1 - 2x_2 + 3x_3 \leq 13 \\
& && 5x_1 - 2x_2 \leq 10 \\
& && x_1, x_2, x_3 \geq 0.
\end{aligned}$$

8.7 Consider the following linear program:

$$\begin{aligned}
\max \quad & 25x_1 + 50x_2 \\
\text{s.t.} \quad & \\
& x_1 + 2x_2 \leq 300 \\
& 2x_1 + x_2 \leq 350 \\
& x_1 + x_2 \leq 200 \\
& x_1, x_2 \geq 0.
\end{aligned}$$

- (a)** Identify all basic feasible solutions. For each basic feasible solution, indicate the basic and nonbasic variables.
- (b)** Solve this linear program without using the simplex method. What are the optimal solution(s)?
- (c)** Now solve this linear program using the simplex method. Use Dantzig's rule to choose the entering variable in each iteration and start the algorithm from the extreme point $(0, 0)$.
- (d)** How does the simplex method indicate that there are multiple solutions?

8.8 Consider the linear program

$$\begin{aligned}
\max \quad & 10x - y \\
\text{s.t.} \quad & \\
& -5x + 3y \leq 15 \\
& 3x - 5y \leq 8 \\
& x, y \geq 0
\end{aligned}$$

- (a)** Convert to canonical form.
- (b)** Use the simplex method to show that the problem is unbounded.

8.9 Consider the following knapsack problem:

$$\begin{aligned}
\max \quad & 10x_1 + 6x_2 + 8x_3 + 12x_4 + 15x_5 + 9x_6 \\
\text{s.t.} \quad & \\
& x_1 + x_2 + 2x_3 + 4x_4 + 6x_5 + 4x_6 \leq 10 \\
& x_k \geq 0, \quad k \in \{1, \dots, 6\}.
\end{aligned}$$

- (a)** Solve this problem using the simplex method, employing Dantzig's rule to select an entering variable.
- (b)** Now consider the general knapsack problem

$$\begin{aligned}
\max \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\text{s.t.} \quad & a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b \\
& x_k \geq 0, \quad k \in \{1, \dots, n\},
\end{aligned}$$

where each $c_k > 0$ and $0 < a_k \leq b$. Develop and prove a simple rule that describes the optimal basic feasible solution.

8.10 Consider the linear program

$$\begin{aligned}
\max \quad & 2x_1 + x_2 + 9x_3 \\
\text{s.t.} \quad & x_1 + x_2 \leq 18 \\
& -2x_1 + x_3 = -12 \\
& 3x_2 + 5x_3 \geq 15 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}$$

(a) Construct the Phase I linear program.

(b) Solve the Phase I linear program to either identify a basic feasible solution to the original problem or show that the problem is infeasible.

(c) Solve the Phase II problem using the solution generated by the Phase I problem as the initial basic feasible solution.

8.11 Consider the linear program

$$\begin{aligned}
\min \quad & 7w_1 - 9w_2 + 4w_3 \\
\text{s.t.} \quad & w_1 - 18w_2 \leq 9 \\
& w_3 + w_4 \geq 14 \\
& w_1 + w_2 - 2w_3 - 3w_4 = 1 \\
& w_1, w_2, w_3, w_4 \geq 0
\end{aligned}$$

(a) Construct the Phase I linear program.

(b) Solve the Phase I linear program to either identify a basic feasible solution to the original problem or show that the problem is infeasible.

(c) If the Phase I problem yields a basic feasible solution to our problem, solve the Phase II problem using this generated basic feasible solution.

8.12 Consider the linear program

$$\begin{aligned}
\max \quad & 2x_1 - 7x_2 + 6x_3 + 5x_4 \\
\text{s.t.} \quad & 2x_1 - 3x_2 - 5x_3 - 4x_4 \geq 20 \\
& 7x_1 + 2x_2 + 6x_3 - 2x_4 \leq 35 \\
& 4x_1 + 5x_2 - 3x_3 - 2x_4 \geq 15 \\
& x_1, x_2, x_3, x_4 \geq 0.
\end{aligned}$$

(a) Construct the Phase I linear program.

(b) Solve the Phase I linear program to either identify a basic feasible solution to the original problem or show that the problem is infeasible.

8.13 Consider the linear program

$$\begin{aligned}
\min \quad & 3x_1 + 6x_2 + 2x_3 \\
\text{s.t.} \quad & x_1 + 3x_2 + 2x_3 \geq 6 \\
& 2x_1 + x_2 + x_3 \geq 3 \\
& x_1, x_2, x_3 \geq 0.
\end{aligned}$$

(a) Construct and solve the Phase I problem, using Dantzig's rule to determine an entering variable. Your optimal Phase I solution should contain a basic artificial variable whose value is 0.

(b) Remove the artificial variable from the basis in (a) before beginning Phase II, then complete Phase II to determine the optimal solution to the original problem.

8.14 Use the two-phase method to solve the linear program

$$\begin{aligned}
\max \quad & x_1 + 3x_2 \\
\text{s.t.} \quad & x_1 + x_2 \geq 3 \\
& x_1 - x_2 \geq 1 \\
& x_1 + 2x_2 \leq 4 \\
& x_1, x_2 \geq 0.
\end{aligned}$$

Graph the two-dimensional feasible region and identify the path of basic (feasible) solutions obtained during each iteration of the method.

8.15 Solve the problems in Exercises 8.1–8.6 using the simplex method (Algorithm 8.2).

8.16 The simplex method is a typically efficient algorithm for solving linear programs. However, there is no version of it that has been shown theoretically to solve any linear program in polynomial time. In fact, it is

much worse. In 1972, Klee and Minty came up with an example in n dimensions that requires $2n - 1$ iterations if Dantzig's rule is used to choose entering variables. These problems are of the form

$$\begin{aligned} \max \quad & \sum_{j=1}^n 10^{n-j} x_j \\ \text{s.t.} \quad & 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1}, \quad i \in \{1, 2, \dots, n\} \\ & x_j \geq 0, \quad j \in \{1, 2, \dots, n\}. \end{aligned}$$

Use the simplex method to solve the following three-dimensional Klee–Minty Problem, where you choose the entering variable as the one with largest reduced cost. Use the basic feasible solution corresponding to $x_1 = x_2 = x_3 = 0$ as your initial solution. It should take exactly seven iterations.

$$\begin{aligned} \max \quad & 100x_1 + 10x_2 + x_3 \\ \text{s.t.} \quad & x_1 \leq 1 \\ & 20x_1 + x_2 \leq 100 \\ & 200x_1 + 20x_2 + x_3 \leq 10,000 \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

8.17 Given the following linear program,

$$\begin{aligned} \max \quad & \frac{3}{4}x_1 - 150x_2 + \frac{1}{50}x_3 - 6x_4 \\ \text{s.t.} \quad & \frac{1}{4}x_1 - 60x_2 - \frac{1}{25}x_3 + 9x_4 \leq 0 \\ & \frac{1}{2}x_1 - 90x_2 - \frac{1}{50}x_3 + 3x_4 \leq 0 \\ & x_3 \leq 1 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

(a) Show that the linear program cycles at a degenerate extreme point if Dantzig's rule is used to choose the entering variable and Bland's rule to choose the leaving variable. Use the simplex method to do your calculations.

(b) Use Bland's rule (for selection of both entering and leaving variables) to solve this problem.

8.18 Consider the linear program

$$\begin{aligned} \max \quad & 2x + 3y \\ \text{s.t.} \quad & x + 2y \leq 11 \\ & 3x + 5y \leq 26 \\ & 0 \leq x \leq 5 \\ & 0 \leq y \leq 4. \end{aligned}$$

(a) Plot the feasible region.

(b) For the problem in canonical form, identify each extended basic feasible solution by indicating the appropriate variable partition $(\mathcal{B}, \mathcal{L}, \mathcal{U})$.

8.19 Consider the linear program

$$\begin{aligned} \max \quad & 4x + 5y \\ \text{s.t.} \quad & x - y \leq 4 \\ & 2x - y \geq -1 \\ & 0 \leq x \leq 6 \\ & 0 \leq y \leq 2. \end{aligned}$$

(a) Plot the feasible region.

(b) For the problem in canonical form, identify each extended basic feasible solution by indicating the appropriate variable partition $(\mathcal{B}, \mathcal{L}, \mathcal{U})$.

8.20 Solve the following linear program using the bounded simplex method (Algorithm 8.3):

$$\begin{aligned} \max \quad & 2x + 3y \\ \text{s.t.} \quad & x + 2y \leq 11 \\ & 3x + 5y \leq 26 \\ & 0 \leq x \leq 5 \\ & 0 \leq y \leq 4. \end{aligned}$$

8.21 Solve the following linear program using the bounded simplex method (Algorithm 8.3):

$$\begin{aligned} \max \quad & 4x + 5y \\ \text{s.t.} \quad & x - y \leq 4 \\ & 2x - y \geq -1 \\ & 0 \leq x \leq 6 \\ & 0 \leq y \leq 2. \end{aligned}$$

8.22 Solve the following linear program using the bounded simplex method (Algorithm 8.3):

$$\begin{aligned} \max \quad & 4x_1 - x_2 + 2x_3 \\ \text{s.t.} \quad & 6x_1 - x_2 + 2x_3 \leq 25 \\ & 3x_1 + x_2 + 5x_3 \leq 40 \\ & 0 \leq x_1 \leq 4 \\ & 0 \leq x_2 \leq 10 \\ & 0 \leq x_3 \leq 9. \end{aligned}$$

8.23 Suppose that in our initial linear program, we have a variable x that is unrestricted in sign. When converting the linear program to canonical form, we replace it with $x = x^+ - x^-$. Is it true that in any optimal solution found by the simplex method, at least one of the variables x^+, x^- has value 0? If so, explain why. If this is not true, provide a counterexample.

8.24 Suppose that during iteration t of the simplex method variable x_j is the leaving variable. Show that x_j cannot be the entering variable in iteration $t + 1$.

8.25 The two-phase method requires us to solve two separate linear programs before obtaining an optimal solution—one problem to find a basic feasible solution, and the other to solve our original problem to optimality. When an initial basic feasible solution is not apparent, we can solve our problem using only one linear program by penalizing the artificial variables by solving large negative number $-M$ ($M > 0$) in the objective function. For example, the linear program

$$\begin{aligned} \max \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & x_1 + x_2 \geq 3 \\ & x_1 - x_2 \geq 1 \\ & x_1 + 2x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

can be rewritten as

$$\begin{aligned}
\max \quad & x_1 + 3x_2 - Ma_1 - Ma_2 \\
\text{s.t.} \quad & \\
& x_1 + x_2 - s_1 + a_1 = 3 \\
& x_1 - x_2 - s_2 + a_2 = 1 \\
& x_1 + 2x_2 + s_3 + a_3 = 4 \\
& x_1, x_2, s_1, s_2, s_3, a_1, a_2 \geq 0,
\end{aligned}$$

where M is a large positive number. If M is large enough, then any infeasible solution will have negative objective value, while every feasible solution will have nonnegative value. We can then use the simplex method to solve this problem, keeping M as a parameter throughout. This method is called the *Big-M Method*. Solve this problem using the Big-M method.

8.26 Given a linear program in canonical form (8.1), let $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ be a basic feasible solution and let \mathbf{d} be a feasible direction at \mathbf{x} . If $\mathbf{d}^N(1), \mathbf{d}^N(2), \dots, \mathbf{d}^N(k)$ are the simplex directions corresponding to the variables x_N , show that

$$\mathbf{d} = d_{N(1)}\mathbf{d}^{N(1)} + d_{N(2)}\mathbf{d}^{N(2)} + \dots + d_{N(k)}\mathbf{d}^{N(k)},$$

where $d_{N(j)}$ is the component of \mathbf{d} corresponding to direction \mathbf{d} . (*Hint:* We know that $A\mathbf{d} = \mathbf{0}$).

8.27 Suppose that during an iteration of the simplex method we are at a nondegenerate basic feasible solution $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ and that the simplex direction \mathbf{d}^k is improving and not an unbounded direction. If we use the Ratio Test to determine our step size, prove that our new solution $\mathbf{x} + \lambda_{\max}\mathbf{d}^k$ is a basic feasible solution.

8.28 Suppose our linear program is

$$\begin{aligned}
\max \quad & \mathbf{c}^T \mathbf{x} \\
\text{s.t.} \quad & \\
& A\mathbf{x} = \mathbf{b} \\
& \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},
\end{aligned}$$

where $\mathbf{l} \neq \mathbf{0}$ and both \mathbf{l}, \mathbf{u} have finite components. Redesign the bounded simplex method to handle this problem and illustrate your method on the linear program

$$\max \quad 2x + 3y$$

s.t.

$$x + 2y \leq 11$$

$$3x + 5y \leq 26$$

$$2 \leq x \leq 5$$

$$1 \leq y \leq 4.$$

CHAPTER 9

LINEAR PROGRAMMING

DUALITY

Now that we've designed an algorithm to solve linear programs, you may be wondering what else there is to study? While solving linear programs is important, there is much more going on theoretically with linear programs, and even with the simplex method itself, than meets the eye. For example, we have seen two different optimality conditions related to linear programs: one involved Farkas' lemma, which indicated when no improving feasible direction can be found, while the second was based on reduced costs. Are these conditions related? That is the subject of this chapter.

In this chapter, we explore a linear program related to our original one, and find that it shares properties with ours; in fact, they are strongly related to each other in terms of optimal values and solutions. We discuss the ramifications of these results and hint at some applications that appear in later chapters.

9.1 MOTIVATION: GENERATING BOUNDS

As we saw in Chapter 5, it is often important and useful to generate both lower and upper bounds to the optimal value z^* of an optimization problem. One of these bounds often corresponds to a feasible solution (e.g., for a maximization problem, every feasible solution generates a lower bound to the optimal value), while the other bound is more difficult to obtain. In fact, many algorithms for solving optimization problems rely on being able to obtain both lower and upper bounds; for integer programs, we will explore

such approaches in Chapters 13 and 14. To begin our study into the theory of linear programs, let's try to quickly obtain an upper bound to a maximization linear program.

■ EXAMPLE 9.1

Suppose we are given the following linear program:

$$z^* = \max \quad 3x_1 + 4x_2 + 8x_3 + 5x_4$$

s.t.

$$2x_1 + 5x_2 + 6x_3 + 3x_4 \leq 30$$

$$6x_1 - x_2 + 3x_3 + 7x_4 \leq 18$$

$$x_1 + 6x_2 - 4x_3 + x_4 \leq 15$$

$$(9.1) \quad x_1, x_2, x_3, x_4 \geq 0.$$

Without solving the linear program, we want to bound its optimal value z^* . For example, since the solution $\mathbf{x}^0 = (1, 1, 1, 1)$ is feasible, we know that $z^* = \mathbf{c}^T \mathbf{x}^0 \geq 20$. Also, since $(1, 0, 3, 0)$ is feasible, we have $z^* \geq 27$. We could continue to do this, but we could never know if a given solution is optimal or not simply by trying to identify feasible solutions randomly. This section, however, is primarily about finding upper bounds to linear programs. For example, we can show that $z^* \leq 50$. How? If we multiply the first constraint by $\frac{5}{3}$, we get

$$(9.2) \quad \frac{10}{3}x_1 + \frac{25}{3}x_2 + 10x_3 + 5x_4 \leq 30 \left(\frac{5}{3} \right) = 50.$$

Any feasible solution to (9.1) must also satisfy (9.2), and since all variables are nonnegative, by comparing (9.2) to our objective, we have

$$3x_1 + 4x_2 + 8x_3 + 5x_4 \leq \frac{10}{3}x_1 + \frac{25}{3}x_2 + 10x_3 + 5x_4 \leq 50.$$

Since this also holds for the optimal solution, we have our bound. Next, we can show that $z^* \leq 48$ because if we add the first and second constraints together, we get

$$(9.3) \quad \begin{aligned} & (2x_1 + 5x_2 + 6x_3 + 3x_4) + (6x_1 - x_2 + 3x_3 + 7x_4) \\ &= 8x_1 + 4x_2 + 9x_3 + 10x_4 \\ &\leq 48, \end{aligned}$$

and the coefficients of each variable in this new constraint are at least the coefficients in the objective function. Thus, any feasible solution must satisfy (9.3), and hence our result follows.

The above example suggests taking (positive) linear combinations of the constraints so that the generated coefficient of each variable is not smaller than the objective coefficient. We ensure that the combinations are positive so that the inequalities remain “ \leq ”, and hence generate an upper bound. To this end, let y_1, y_2, y_3 denote the nonnegative amounts by which we multiply constraints of (9.1). Doing this gives the following inequality:

$$(2y_1 + 6y_2 + y_3)x_1 + (5y_1 - y_2 + 6y_3)x_2 \\ +(6y_1 + 3y_2 - 4y_3)x_3 + (3y_1 + 7y_2 + y_3)x_4 \leq 30y_1 + 18y_2 + 15y_3.$$

One way to ensure that $30y_1 + 18y_2 + 15y_3$ is an upper bound on z^* is to demand that

$$\begin{aligned} 2y_1 + 6y_2 + y_3 &\geq 3 \\ 5y_1 - y_2 + 6y_3 &\geq 4 \\ 6y_1 + 3y_2 - 4y_3 &\geq 8 \\ 3y_1 + 7y_2 + y_3 &\geq 5 \\ y_1, y_2, y_3 &\geq 0. \end{aligned}$$

These look like constraints for a linear program! To find the smallest upper bound to z^* using this technique, we solve the linear program

$$\min \quad 30y_1 + 18y_2 + 15y_3$$

s.t.

$$\begin{aligned} 2y_1 + 6y_2 + y_3 &\geq 3 \\ 5y_1 - y_2 + 6y_3 &\geq 4 \\ 6y_1 + 3y_2 - 4y_3 &\geq 8 \\ 3y_1 + 7y_2 + y_3 &\geq 5 \\ y_1, y_2, y_3 &\geq 0. \end{aligned}$$

Note that this linear program uses the same “input” as the original linear program, in that none of the numbers has changed, just their positions within the new linear program.

What would happen if one of the constraints of (9.1) changed its “sense”

from “ \leq ” to “ \geq ” or equality? For example, suppose the first constraint becomes

$$2x_1 + 5x_2 + 6x_3 + 3x_4 \geq 30.$$

In order to generate an upper bound on z^* in a similar fashion, we would need to multiply this constraint by the nonpositive variable y_1 in order to switch the inequality to “ \leq ” and generate an upper bound. If, for example, the third constraint becomes the equality constraint

$$x_1 + 6x_2 - 4x_3 + x_4 = 15,$$

we can multiply this constraint by any (positive, negative, or zero) number y_3 and still construct an upper bound. Thus, by altering the sign of the multiplier y_k , we can generate an upper bound irrespective of the type of linear constraint in our problem.

Now let’s consider different variable types. In Example 9.1, in order to generate an upper bound on z^* , since each variable is nonnegative, we wanted to ensure that each constructed objective coefficient is at least as large as the original coefficient. What would happen if, say, $x_2 \leq 0$? Its constructed objective coefficient, using the constraint multipliers y_1 , y_2 , and y_3 , is still $5y_1 - y_2 + 6y_3$. In order to maintain the structure of our upper-bound technique, we would need to ensure that

$$(5y_1 - y_2 + 6y_3)x_2 \geq 4x_2$$

for all feasible values of x_2 . Since $x_2 \leq 0$, this implies that

$$5y_1 - y_2 + 6y_3 \leq 4.$$

Suppose, we have x_2 that is unrestricted in sign, so that it can be positive, negative, or zero? We would again want to maintain $(5y_1 - y_2 + 6y_3)x_2 \geq 4x_2$ for all values of x_2 , but this can be done only if

$$(5y_1 - y_2 + 6y_3) = 4.$$

Thus, we can alter this upper-bounding approach to handle any type of variable bounds.

■ EXAMPLE 9.2

Let’s construct a linear program whose optimal value is an upper bound to

the optimal value z^* of the linear program

$$\max \quad 8x_1 + 3x_2 + x_4$$

s.t.

$$3x_1 + 2x_2 + 3x_3 + 5x_4 \leq 14$$

$$x_1 + x_3 - x_4 \geq 2$$

$$5x_1 + 3x_2 + 2x_3 + x_4 = 25$$

$$(9.4) \quad x_1, x_2 \geq 0, x_3 \leq 0, x_4 \text{ unrestricted.}$$

Let y_1 , y_2 , and y_3 be the multipliers for the constraints in (9.4). Based on our above discussion, we must have the variable bounds $y_1 \geq 0$, $y_2 \leq 0$, and y_3 unrestricted in sign. Furthermore, based upon the bounds of the variables x_1 , x_2 , x_3 , x_4 , we would have the following constraints on the values of y_1 , y_2 , and y_3 :

$$3y_1 + y_2 + 5y_3 \geq 8$$

$$2y_1 + 3y_3 \geq 3$$

$$3y_1 + y_2 + 2y_3 \leq 0$$

$$5y_1 - y_2 + y_3 = 1.$$

If y_1 , y_2 , and y_3 satisfy these constraints and bounds, an upper bound to z^* would be $z^* \leq 14y_1 + 2y_2 + 25y_3$. Thus, the smallest upper bound to z^* using this technique can be found by solving the linear program

$$\min \quad 14y_1 + 2y_2 + 25y_3$$

s.t.

$$3y_1 + y_2 + 5y_3 \geq 8$$

$$2y_1 + 3y_3 \geq 3$$

$$3y_1 + y_2 + 2y_3 \leq 0$$

$$5y_1 - y_2 + y_3 = 1$$

$$y_1 \geq 0, y_2 \leq 0, y_3 \text{ unrestricted.}$$

9.2 DUAL LINEAR PROGRAM

In each example in Section 9.1, we saw very similar things: each started with

a (maximization) linear program with objective coefficients \mathbf{c} , constraint matrix A , and right-hand side vector \mathbf{b} and ended with a (minimization) linear program with objective coefficients \mathbf{b} , constraint matrix A^T , and right-hand side vector \mathbf{c} . Clearly, something more formal is happening here. This section, and in fact the rest of this chapter, explores this new linear program. To do this, we start with a linear program in standard form.

Dual Linear Program Given a linear program in standard form

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{9.5}$$

the associated linear program

$$\begin{aligned} \min \quad & \mathbf{b}^T \mathbf{y} \\ \text{s.t.} \quad & \\ & A^T \mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{9.6}$$

is known as the **dual linear program** or just the **dual**.

Primal Linear Program The original linear program (9.5) is often known as the **primal linear program** or just the **primal**.

With each constraint in the primal linear program we associate a **dual variable**, and with each variable in the primal we associate a constraint in the dual linear program. Note that, by the rationale of Example 9.1, solving the dual linear program (9.6) yields an upper bound to the optimal solution of (9.5); we formally prove this result in the next section. While this is a nice fact given linear programs in standard form, it doesn't help us for linear programs in any other form. However, a linear program does not have to be in standard form in order to have an associated dual linear program.

■ EXAMPLE 9.3

Consider again the linear program from Example 9.2.

$$\begin{aligned}
\max \quad & 8x_1 + 3x_2 + x_4 \\
\text{s.t.} \quad & \\
& 3x_1 + 2x_2 + 3x_3 + 5x_4 \leq 14 \\
& x_1 + x_3 - x_4 \geq 2 \\
& 5x_1 + 3x_2 + 2x_3 + x_4 = 25 \\
& x_1, x_2 \geq 0 \\
& x_3 \leq 0 \\
(9.7) \quad & x_4 \text{ unrestricted.}
\end{aligned}$$

Since we only know how to find the dual linear program of a primal linear program in standard form, to find the dual of our linear program we first must convert this linear program into standard form. This is done by multiplying the second constraint by -1 , converting the third constraint into two \leq -constraints, substituting $x'_3 = -x_3 \geq 0$, and adding the relationship $x_4 = x_4^+ - x_4^-$, where $x_4^+, x_4^- \geq 0$.

$$\begin{aligned}
\max \quad & 8x_1 + 3x_2 + (x_4^+ - x_4^-) \\
\text{s.t.} \quad & \\
& 3x_1 + 2x_2 - 3x'_3 + 5(x_4^+ - x_4^-) \leq 14 \\
& -x_1 + x'_3 + (x_4^+ - x_4^-) \leq -2 \\
& 5x_1 + 3x_2 - 2x'_3 + (x_4^+ - x_4^-) \leq 25 \\
& -5x_1 - 3x_2 + 2x'_3 - (x_4^+ - x_4^-) \leq -25 \\
(9.8) \quad & x_1, x_2, x'_3, x_4^+, x_4^- \geq 0.
\end{aligned}$$

Now, associating dual variable y_1, y'_2, y_3^+, y_3^- with each constraint in (9.8), the dual is

$$\begin{aligned}
\min \quad & 14y_1 - 2y'_2 + 25y_3^+ - 25y_3^- \\
(9.9) \quad \text{s.t.} \quad &
\end{aligned}$$

$$\begin{aligned}
3y_1 - y'_2 + 5y_3^+ - 5y_3^- &\geq 8 \\
2y_1 + 3y_3^+ - 3y_3^- &\geq 3 \\
-3y_1 + y'_2 - 2y_3^+ + 2y_3^- &\geq 0 \\
5y_1 + y'_2 + y_3^+ - y_3^- &\geq 1 \\
-5y_1 - y'_2 - y_3^+ + y_3^- &\geq -1 \\
y_1, y'_2, y_3^+, y_3^- &\geq 0.
\end{aligned}$$

While this is the dual linear program, it does not have the nice features of the objective coefficients being the right-hand side values of the original linear program, or the right-hand side values being the objective coefficients of the original. Can we transform (9.9) into an equivalent form so as to preserve these features? If we rename $y_2 = -y'_2$, the objective coefficient of y_2 would be 2, which is the right-hand side of the second constraint in our original linear program (9.7). Of course, $y_2 \leq 0$. Similarly, if we use the substitution, $y_3 = y_3^+ - y_3^-$, where y_3 is unrestricted in sign, then the coefficient of y_3 is 25, which is the right-hand side of the third constraint in (9.7). In addition, if we multiply the third constraint by -1 and combine the last two constraints into one equality constraint, we get the following (transformed) dual linear program to (9.7):

$$\begin{aligned}
\text{min} \quad & 14y_1 + 2y_2 + 25y_3 \\
\text{s.t.} \quad & \\
& 3y_1 + y_2 + 5y_3 \geq 8 \\
& 2y_1 + 3y_3 \geq 3 \\
& 3y_1 + y_2 + 2y_3 \leq 0 \\
& 5y_1 - y_2 + y_3 = 1 \\
& y_1 \geq 0, y_2 \leq 0 \\
(9.10) \quad & y_3 \text{ unrestricted in sign.}
\end{aligned}$$

Note that this is exactly the linear program we found in Example 9.2.

What this example illustrates is the following notion.

Every linear program has an associated dual linear program.

What if we take the dual of our dual problem?

■ EXAMPLE 9.4

Suppose we were to start with the following linear program in standard form:

$$\begin{aligned} \max \quad & 8x_1 + 4x_2 + 3x_3 \\ \text{s.t.} \quad & 3x_1 + 2x_2 + 6x_3 \leq 15 \\ & 4x_1 - 2x_2 + 3x_3 \leq 10 \\ (9.11) \quad & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Its dual is (with dual variables y_1, y_2 associated with the constraints)

$$\begin{aligned} \min \quad & 15y_1 + 10y_2 \\ \text{s.t.} \quad & 3y_1 + 4y_2 \geq 8 \\ & 2y_1 - 2y_2 \geq 4 \\ & 6y_1 + 3y_2 \geq 3 \\ (9.12) \quad & y_1, y_2 \geq 0. \end{aligned}$$

What would the dual of (9.12) be? To answer this, let us first convert (9.12) into standard form by (1) transforming each constraint into \leq -constraints and (2) making the minimization problem a maximization problem. To do (1), we just multiply each constraint by (-1) . To do (2), recall that $\min f(x) = -\max [-f(x)]$. Therefore, our transformed linear program would be

$$\begin{aligned} -\max \quad & -15y_1 - 10y_2 \\ \text{s.t.} \quad & -3y_1 - 4y_2 \leq -8 \\ & -2y_1 + 2y_2 \leq -4 \\ & -6y_1 - 3y_2 \leq -3 \\ (9.13) \quad & y_1, y_2 \geq 0. \end{aligned}$$

The dual of (9.13) is then (with dual variables z_1, z_2, z_3 associated with the constraints)

$$\begin{aligned}
& -\min && -8z_1 - 4z_2 - 3z_3 \\
& \text{s.t.} && \\
& && -3z_1 - 2z_2 - 6z_3 \geq -15 \\
& && -4z_1 + 2z_2 - 3z_3 \geq -10 \\
& && z_1, z_2, z_3 \geq 0.
\end{aligned}$$

If we again transform the minimization problem into a maximization problem, and multiply each constraint by (-1) , we get

$$\begin{aligned}
& \max && 8z_1 + 4z_2 + 3z_3 \\
& \text{s.t.} && \\
& && 3z_1 + 2z_2 + 6z_3 \leq 15 \\
& && 4z_1 - 2z_2 + 3z_3 \leq 10 \\
& && z_1, z_2, z_3 \geq 0,
\end{aligned}$$

which is the same linear program as (9.11).

This important concept from Example 9.4 is given formally below.

Given any (primal) linear program P and its dual D , then the dual of D is again the original primal P .

In fact, we can come up with rules that govern how to generate the dual of a given linear program. The basic rules that apply to all linear programs are as follows:

1. The right-hand side values of the constraints of the primal become the objective function coefficients of the dual.
2. The objective function coefficients of the primal become the right-hand side values of the dual.
3. The constraint matrix of the primal becomes transposed in the dual.
4. Minimization primal problems become maximization dual problems, and vice versa.
5. The “sense” of each constraint (i.e., \leq , \geq , or $=$) in the primal determines the “sense” of each variable in the dual (i.e., ≥ 0 , ≤ 0 , or unrestricted in sign).
6. The “sense” of each variable (i.e., ≥ 0 , ≤ 0 , or unrestricted in sign) in the primal determines the “sense” of each constraint (i.e., \leq , \geq , or $=$) in the

dual.

To Construct the Dual of any given Linear Program:

1. Assign to each constraint in the primal a corresponding dual variable y_i .
2. Write down the right-hand side coefficients as coefficients of the dual variables in the objective function.
3. Write the primal objective function coefficients as the right-hand side of the constraints.
4. Transpose the constraint matrix A by looking at the coefficients columnwise for each primal variable.
5. Find the “sense” of each constraint and each dual variable using the following:

Max Problem	\leftrightarrow	Min Problem
\leq constraint	\leftrightarrow	$y_i \geq 0$
\geq constraint	\leftrightarrow	$y_i \leq 0$
$=$ constraint	\leftrightarrow	y_i unrestricted
$x_i \geq 0$	\leftrightarrow	\geq constraint
$x_i \leq 0$	\leftrightarrow	\leq constraint
x_i unrestricted	\leftrightarrow	$=$ constraint

■ EXAMPLE 9.5

To find the dual of the following linear program:

$$\min \quad 3x_1 - 2x_2 + x_3$$

s.t.

$$x_1 + x_2 + 3x_3 \leq 8$$

$$2x_2 - x_3 \geq 6$$

$$x_1 + x_3 = 4$$

x_1 unrestricted

$x_2 \geq 0$

$x_3 \leq 0$,

associate with each constraint the variables y_i , $i = 1, \dots, 3$. The “shell” of the dual linear program would then be

$$\max \quad 8y_1 + 6y_2 + 4y_3$$

s.t.

$$y_1 + y_3 ? 3$$

$$y_1 + 2y_2 ? -2$$

$$3y_1 - y_2 + y_3 ? 1,$$

where “?” indicates that we still need to determine the sense of each dual constraint. Using the rules above, we see that the first constraint would be “=” since x_1 is unrestricted in the primal; the second constraint is “ \leq ” since $x_2 \geq 0$; and the third constraint would be “ \geq ” since $x_3 \leq 0$. Similarly, $y_1 \leq 0$ since the first primal constraint is “ \leq ” in a minimization problem; $y_2 \geq 0$ since the second primal constraint is “ \geq ” in a minimization problem; and y_3 is unrestricted since the third primal constraint is “=”. Hence, our dual is

$$\max \quad 8y_1 + 6y_2 + 4y_3$$

s.t.

$$y_1 + y_3 = 3$$

$$y_1 + 2y_2 \leq -2$$

$$3y_1 - y_2 + y_3 \geq 1$$

$$y_1 \leq 0$$

$$y_2 \geq 0$$

$$y_3 \text{ unrestricted.}$$

It is useful to note that we first established the dual linear program as a way of generating upper bounds on our primal linear program. In fact, with this approach in mind, we can generate dual problems for any optimization problem, such as nonlinear programs and integer programs; we touch on this notion in Section 9.7. While we will not be addressing these issues here, it’s important for us to recognize that this is a more general (and useful) concept in optimization.

9.3 DUALITY THEOREMS

In Example 9.1, we constructed a linear program, which turned into the dual,

that enabled us to compute an upper bound to another linear program. We can formalize this relationship for any linear program in standard form. This relationship is known as **Weak Duality**, and it is the basis for the dual problem being formed for any optimization problem, regardless of it being a linear program or not.

Theorem 9.1 (Weak Duality) *Let P denote the linear program in standard form (9.5) with objective function $\mathbf{c}^T \mathbf{x}$ and D denote its dual (9.6) with objective function $\mathbf{b}^T \mathbf{y}$. Let \mathbf{x}^0 denote a feasible solution to P and \mathbf{y}^0 denote a feasible solution to D . Then,*

$$\mathbf{c}^T \mathbf{x}^0 \leq \mathbf{b}^T \mathbf{y}^0.$$

Proof Since \mathbf{x}^0 is a feasible solution to P , we know that

$$A\mathbf{x}^0 \leq \mathbf{b}.$$

If we multiply each side by the vector \mathbf{y}^0 , the inequality does not change (since $\mathbf{y}^0 \geq \mathbf{0}$), and we get $(\mathbf{y}^0)^T A\mathbf{x}^0 \leq (\mathbf{y}^0)^T \mathbf{b}$. Taking the transpose of each side gives

$$(\mathbf{x}^0)^T A^T \mathbf{y}^0 \leq \mathbf{b}^T \mathbf{y}^0.$$

Since \mathbf{y}^0 is feasible to D , we know $A^T \mathbf{y}^0 \geq \mathbf{c}$. Therefore, since $\mathbf{x}^0 \geq \mathbf{0}$, we have

$$\mathbf{b}^T \mathbf{y}^0 \geq (\mathbf{x}^0)^T A^T \mathbf{y}^0 \geq (\mathbf{x}^0)^T \mathbf{c} = \mathbf{c}^T \mathbf{x}^0.$$

Corollary 9.1 *Let P denote any maximization primal linear program and D its corresponding dual. If P has a feasible solution with value z_P and D has a feasible solution with value z_D , then*

$$z_P \leq z_D.$$

What the weak duality theorem and its corollary are saying is that, given a linear program and its dual, any feasible solution to the maximization problem has an objective function value that is not more than the value of any feasible solution to the minimization problem. In fact, the weak duality theorem also implies the following corollaries, whose proofs are left as exercises.

Corollary 9.2 *Let P denote the primal linear program with objective function $\mathbf{c}^T \mathbf{x}$ and D its corresponding dual with objective function $\mathbf{b}^T \mathbf{y}$. If \mathbf{x}^0 and \mathbf{y}^0 are feasible solutions to P and D , respectively, and*

$$\mathbf{c}^T \mathbf{x}^0 = \mathbf{b}^T \mathbf{y}^0,$$

then \mathbf{x}^0 and \mathbf{y}^0 are optimal solutions to P and D , respectively.

Corollary 9.3 *If the primal linear program is unbounded, then its dual linear program must be infeasible.*

Corollary 9.4 *If the dual linear program is unbounded, then the primal linear program must be infeasible.*

What happens if either the primal or dual is infeasible? Does it necessarily mean that the other linear program is unbounded? Not necessarily, for it is possible that both linear programs are infeasible.

■ EXAMPLE 9.6

The following primal–dual pair of linear programs are both infeasible:

$$\begin{array}{ll} \max & x_2 \\ \text{s.t.} & \\ & x_1 \leq -1 \\ & x_2 \geq -1 \\ & x_1, x_2 \geq 0, \end{array} \quad \begin{array}{ll} \min & -y_1 + y_2 \\ \text{s.t.} & \\ & y_1 \geq 0 \\ & y_2 \geq 1 \\ & y_1 \geq 0, y_2 \leq 0. \end{array}$$

In Corollary 9.2, we saw that if both the primal and the dual linear programs have feasible solutions with equal objective function values, then those solutions must be optimal to their respective linear programs. How often does it occur that the primal and dual optimal solutions have equal values? For example, it is not difficult to check that each of the (feasible) primal–dual pairs that we have seen throughout this chapter have equal optimal values. What is amazing is that this always occurs (provided both have optimal solutions). This fact is often referred to as **Strong Duality**.

Theorem 9.2 (Strong Duality Theorem) *Let P denote the primal linear program and D its dual.*

- (a) *If P has a finite optimal solution, then D also has a finite optimal solution with the same objective function value.*
- (b) *If P and D both have feasible solutions, then*
 - *P has a finite optimal solution \mathbf{x}^* ;*
 - *D has a finite optimal solution \mathbf{y}^* ;*
 - *the optimal values of P and D are equal.*

Proof Throughout, we assume that P is in canonical form

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

and, therefore, D is of the form

$$\begin{aligned} \min \quad & \mathbf{b}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{A}^T \mathbf{y} \geq \mathbf{c}. \end{aligned}$$

Let \mathbf{x}^* be an optimal basic feasible solution to P (why does this exist?). We can decompose \mathbf{x}^* into its basic and nonbasic components \mathbf{x}_B^* and \mathbf{x}_N^* , giving

$$\mathbf{x}_B^* = \mathbf{B}^{-1} \mathbf{b}, \quad \mathbf{x}_N^* = \mathbf{0}, \quad z^* = \mathbf{c}_B^T \mathbf{x}_B^* = \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b},$$

where B is the submatrix of A whose columns correspond to the basic variables. Let

$$\mathbf{y}^T = \mathbf{c}_B^T \mathbf{B}^{-1}.$$

Then,

$$\mathbf{b}^T \mathbf{y} = \mathbf{y}^T \mathbf{b} = \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}.$$

Therefore, if we can show that \mathbf{y} is feasible to D , then we are done.

Before we do this, note that if we partition A into

$$A = [B : N],$$

the constraints of D become

$$[B : N]^T \mathbf{y} \geq \mathbf{c} \Leftrightarrow \begin{bmatrix} B^T \\ N^T \end{bmatrix} \mathbf{y} \geq \begin{bmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{bmatrix}$$

or equivalently,

$$\begin{aligned} B^T \mathbf{y} &\geq \mathbf{c}_B \\ (9.14) \quad N^T \mathbf{y} &\geq \mathbf{c}_N. \end{aligned}$$

To show that \mathbf{y} satisfies the first set of constraints in (9.14), note that

$$\begin{aligned} \mathbf{0}^T &= \mathbf{c}_B^T - \mathbf{c}_B^T \\ &= \mathbf{c}_B^T - \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{B} \\ &= \mathbf{c}_B^T - \mathbf{y}^T \mathbf{B}. \end{aligned}$$

Therefore, $B^T \mathbf{y} = \mathbf{c}_B$. Next, since \mathbf{x}^* is optimal, this implies that all reduced costs associated with the nonbasic variables are nonpositive. Hence,

$$\begin{aligned}\bar{\mathbf{c}}_N^T &= \mathbf{c}_N^T - \mathbf{c}_B^T B^{-1} N \leq \mathbf{0}^T \\ \implies \mathbf{c}_N^T - \mathbf{y}^T N &\leq \mathbf{0}^T \\ \implies N^T \mathbf{y} &\geq \mathbf{c}_N.\end{aligned}$$

Thus, \mathbf{y} is feasible in D and has the same objective function value as \mathbf{x}^* .

The proof of part (b) is left as an exercise.

Note how powerful the strong duality theorem is. If both the primal and its dual have feasible solutions, then both have optimal solutions with the same objective function value. In other words, two linear programs that are only related by the inputs to the problem (A , \mathbf{c} , and \mathbf{b}) will have equal objective function values if they are both feasible and bounded. This is really an amazing fact, and one that has proved very useful to operations researchers.

However, there is something else the strong duality theorem is saying. Consider the proof of Theorem 9.2. Did you recognize the form of the optimal dual solution obtained? It was, in fact, the vector of simplex multipliers we used in the simplex method to calculate the reduced costs. Also in the proof, we used the fact that the reduced costs of all nonbasic variables are nonpositive to show that the given dual vector \mathbf{y} was feasible. In other words, when we solve a linear program using the simplex method, we are doing much more than we initially thought.

-
- (a) When the simplex method (Algorithm 8.2) is used to solve a linear program P , the vector \mathbf{y} of simplex multipliers obtained in Step 1 is really a potential feasible solution to the corresponding dual problem D . It is constructed so that a portion of the constraints in D is satisfied. These constraints correspond to the basic variables of the current basic feasible solution of P .
 - (b) When we check to see if there is an improving simplex direction to P (via positive reduced costs), we are really checking to see if there is a constraint in D that is not satisfied by \mathbf{y} . Each time we move to a new basic feasible solution, we make at least one previous constraint that was violated by \mathbf{y} active in D .
 - (c) When an optimal basic feasible solution is found for P , an optimal

solution is also found for D . Furthermore, the constraints that are active in D correspond to the basic variables in the optimal basic feasible solution found for P , while the nonactive constraints in D correspond to the nonbasic variables.

(d) Note that the simplex method does not find a feasible dual solution until the last step, when it finds the optimal dual solution, or indicates that the dual problem is infeasible.

Part (c) above also implies is that if a variable in the optimal solution to P is positive, the corresponding constraint in D must be active, and that if a constraint in D is not active, its corresponding variable in P must have value 0. This property extends to any linear program and is known as **Complementary Slackness**.

Theorem 9.3 (Complementary Slackness Theorem) *Let P denote the primal linear program and D its corresponding dual. Suppose P and D both have finite optimal values, and let \mathbf{x}^* and \mathbf{y}^* be optimal solutions to P and D , respectively. Let s_i^* denote the slack of the i th constraint of P and let w_j^* denote the slack of the j th constraint of D at the respective optimal solutions. Then,*

$$x_j^* w_j^* = 0, \quad j \in \{1, 2, \dots, n\}$$

$$y_i^* s_i^* = 0, \quad i \in \{1, 2, \dots, m\}.$$

Proof We shall assume that P is in standard form (9.5). When P is converted to canonical form, its constraints are

$$A\mathbf{x} + I\mathbf{s} = \mathbf{b},$$

where $\mathbf{x}, \mathbf{s} \geq \mathbf{0}$. Suppose that \mathbf{y}^* is the optimal solution to D . Then,

$$\mathbf{y}^{*T} A\mathbf{x} + \mathbf{y}^{*T} I\mathbf{s} = \mathbf{y}^{*T} \mathbf{b} = \mathbf{b}^T \mathbf{y}^*.$$

If \mathbf{x}^* is the optimal solution to P , with optimal slack variables \mathbf{s}^* , then $\mathbf{b}^T \mathbf{y}^* = \mathbf{c}^T \mathbf{x}^*$, implying that

$$\mathbf{c}^T \mathbf{x}^* = \mathbf{b}^T \mathbf{y}^* = \mathbf{y}^{*T} A\mathbf{x}^* + \mathbf{y}^{*T} \mathbf{s}^*.$$

Since \mathbf{y}^* is feasible to D , we know $A^T \mathbf{y}^* \geq \mathbf{c}$, or $\mathbf{y}^{*T} A \geq \mathbf{c}^T$. Therefore,

$$\mathbf{c}^T \mathbf{x}^* = \mathbf{y}^{*T} A\mathbf{x}^* + \mathbf{y}^{*T} \mathbf{s}^* \geq \mathbf{c}^T \mathbf{x}^* + \mathbf{y}^{*T} \mathbf{s}^*,$$

or $\mathbf{y}^{*T} \mathbf{s}^* \leq 0$. Since we assumed that P is in standard form (9.5), then D is in form (9.6), which makes $\mathbf{y}^* \geq \mathbf{0}$. Therefore, since both \mathbf{s}^* and \mathbf{y}^* are

nonnegative vectors, the only way that $\mathbf{y}^{*T} \mathbf{s}^* \leq 0$ is if

$$y_j^* s_j^* = 0$$

for all $j = 1, 2, \dots, m$. Similarly, we can show that $x_j^* w_j^* = 0$, for $j = 1, 2, \dots, n$.

Complementary slackness says that, given *any* linear program P and its dual D , if, at the optimal solution, a variable is positive in $P(D)$, the corresponding constraint in $D(P)$ must be active. Note that it is possible for a variable to have value 0 and its corresponding constraint be active. What is amazing is that, if we look at the simplex method again, we see something very remarkable.

During each iteration of the simplex method, we have a primal feasible solution and “potential” dual solution that satisfy the complementary slackness conditions. In other words, we can view the simplex method as starting with a primal feasible solution that generates a potential dual solution that together satisfies the complementary slackness conditions and searches for a dual feasible solution with the same objective function value.

Complementary slackness and strong duality can also be used as a “certificate of optimality” to show that a given solution to the primal linear program is in fact optimal.

■ EXAMPLE 9.7

In this example, we will use complementary slackness and strong duality to show that the point $(5, 4)$ is optimal to the linear program P :

$$\max \quad 15x + 5y$$

s.t.

$$4x + y \leq 24$$

$$x + 3y \leq 24$$

$$3x + 2y \leq 23$$

$$x, y \geq 0.$$

The dual D to this linear program is

$$\min \quad 24y_1 + 24y_2 + 23y_3$$

s.t.

$$4y_1 + y_2 + 3y_3 \geq 15$$

$$y_1 + 3y_2 + 2y_3 \geq 5$$

$$y_1, y_2, y_3 \geq 0.$$

Note that, in P , we have that $s_1 = s_3 = 0$ and $s_2 = 7$. This implies that $y_2 = 0$, by complementary slackness, and that the dual constraints

$$4y_1 + y_2 + 3y_3 \geq 15$$

$$y_1 + 3y_2 + 2y_3 \geq 5$$

are active. Thus, in the optimal dual solution, we have

$$4y_1 + 3y_3 = 15$$

$$y_1 + 2y_3 = 5.$$

Solving this system of equations gives $y_1 = 3$, $y_3 = 1$, and so our dual solution is $(3, 0, 1)$. Its objective function value is $3(24) + 23 = 95$, which is the same objective value as $(5, 4)$ in P . Hence, $(5, 4)$ is optimal to P and $(3, 0, 1)$ is optimal to D .

9.4 ANOTHER INTERPRETATION OF THE SIMPLEX METHOD

Duality theory and complementary slackness provide the theoretical backbone of linear programming. As noted in the previous section, they generate a “certificate of optimality,” enabling us to determine whether a given feasible solution is optimal. However, this is not the only information they yield. In this section, we reexamine the simplex method and see how it is not just an improving search method.

But why bother with such a re-examination? Often in mathematics a problem is solved by a mathematical construct; in our case, linear programs were solved by a specific version of an improving search method. However, it is not until the problem (and its solution) is explored in a more general

setting that other approaches can be derived and additional theories developed. This is evident, for example, in algebra, where Galois theory and group theory were formulated to solve the problem “how do we solve a polynomial equation in one variable?” It is not until these new insights are developed that we can truly understand the applicability of the original problem.

We noted in the previous section that in each iteration, the simplex method maintained a feasible solution to the primal problem, constructed simplex multipliers that together satisfied the complementary slackness condition, and tried to determine if the simplex multipliers yielded a feasible solution to the dual problem. In this view, the reduced costs of the primal solution corresponded to the slack/surplus variables of the dual constraints. If the slack/surplus variables were positive (for a primal maximization problem), the proposed dual solution was not feasible. Thus, both the primal and the dual solutions needed to be adjusted; the primal solution maintained feasibility while the proposed dual solution tried to achieve feasibility. The simplex method iterated until both the primal and the dual solutions were feasible. Since complementary slackness was maintained for each pair of primal–dual solutions, once both were feasible they were optimal by the strong duality theorem (Theorem 9.2).

With this insight into the simplex method we are now free to consider alternative approaches to solving linear programs. For example, what if we maintained, at every iteration, a feasible solution to the dual problem that satisfied complementary slackness with a proposed primal solution and tried to achieve primal feasibility? There are variations of the simplex method that do just this, including one we will study in Chapter 11, the dual simplex method. In fact, the version of the simplex method that we have studied is often referred to as the primal simplex method because it maintains primal feasibility. Another approach we can take is to maintain primal and dual feasible solutions throughout and strive to achieve complementary slackness; we explore one such approach in Chapter 11.

9.5 FARKAS’ LEMMA

REVISITED

In Section 6.4, we saw that when we are implementing the general improving search method (Algorithm 6.2) on a linear program in standard form, it is possible to determine whether or not an improving feasible direction \mathbf{d} exists at an extreme point by considering whether a related system of equations had a solution. This notion, referred to as Farkas' lemma, is restated below.

Lemma 9.1 (Farkas' Lemma) *Given an $m \times n$ matrix A and n -dimensional vector \mathbf{c} , one and only one of the following two systems has a solution:*

$$\text{System 1: } A\mathbf{d} \leq \mathbf{0} \text{ and } \mathbf{c}^T \mathbf{d} > 0$$

$$\text{System 2: } A^T \mathbf{y} = \mathbf{c} \text{ and } \mathbf{y} \geq \mathbf{0}.$$

Look closely at the two systems. System 1 concerned the existence of an improving feasible direction, where A in this case represented the matrix corresponding to the active constraints. System 2, however, begins to remind us of the constraints for a dual linear program. Does this make sense?

To answer this, let's return to the example problem from Section 6.4:

$$\begin{aligned} \max \quad & 3x + 4y \\ \text{s.t.} \quad & \\ & x + y \leq 8 \\ & 2x + y \leq 10 \\ & x \leq 4 \\ (9.15) \quad & x, y \geq 0. \end{aligned}$$

We saw that at the extreme point $(0, 8)$, there are no improving feasible directions. This is because all feasible directions must satisfy

$$\begin{aligned} d_x + d_y &\leq 0 \\ -d_x &\leq 0, \end{aligned}$$

and to be improving it must satisfy $3d_x + 4d_y > 0$. This corresponds to the system

$$\begin{aligned} \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} &\leq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} &> 0; \end{aligned}$$

thus, System 1 in Farkas' lemma has no solution. Now consider System 2:

$$\begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}^T \begin{bmatrix} w_1 \\ w_x \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_x \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} w_1 \\ w_x \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

We chose the subscripts of the variables w_1 , w_x to indicate which of the original constraints/bounds are active at $(0, 8)$. Note that $w_1 = 4$, $w_x = 1$ is a solution to this system.

Now let's return to the original linear program (9.15). Its optimal solution is $(x, y) = (0, 8)$ with value 32, and its dual problem is

$$\begin{aligned} \min \quad & 8w_1 + 10w_2 + 4w_3 \\ \text{s.t.} \quad & \end{aligned}$$

$$\begin{aligned} w_1 + 2w_2 + w_3 &\geq 3 \\ w_1 + w_2 &\geq 4 \\ w_1, w_2, w_3 &\geq 0, \end{aligned}$$

Its optimal solution is $(w_1, w_2, w_3) = (4, 0, 0)$ with value 32. In this dual optimal solution and in the solution to System 2 for Farkas' lemma, we had $w_1 = 4$. In fact, since constraints 2 and 3 from our primal problem were not active, we could have thought of their "Farkas multipliers" as having value 0, much like the dual values. In other words, the solution to System 2 from Farkas' lemma, at an extreme point for which there is no improving feasible direction, can be transformed into a solution to the dual linear program that is (a) feasible and (b) has the same objective value as the feasible primal solution. But this is exactly what the strong duality theorem says must occur! In essence, we have verified the strong duality theorem for this particular problem using the results from Farkas' lemma.

It should not be too hard to imagine that this notion can be formalized. In fact, we can mimic the above argument to prove the strong duality theorem for a linear program in standard form using Farkas' lemma (see Exercise 9.11). Also, since there are many variants of Farkas' lemma, depending on the particular System 1 (say having $\mathbf{A}\mathbf{d} = \mathbf{0}$, for instance), the strong duality theorem associated with any form of a linear program can be proven using Farkas' lemma. Finally, we should note that we can prove Farkas' lemma using strong duality theorem, indicating a type of equivalence between these results (see Exercise 9.12).

It is useful to recall that Farkas' lemma is a result of the existence of a

solution to a given system of linear equations and inequalities and not a result directly pertaining to linear programming. However, Farkas' lemma can be viewed as another form of the strong duality theorem, and hence provides a certificate of optimality for our linear programming solution.

9.6 ECONOMIC INTERPRETATION OF THE DUAL

What is remarkable about the dual program is that not only does it help certify the optimality of a solution to the primal problem but its variables and constraints do also have their own interpretation and insights within the context of the primal problem. While this interpretation will vary from model to model, we can see the general application through an example.

■ EXAMPLE 9.8

Wood Built Furniture (WBF) is a small furniture shop that produces three types of bookshelves: models A, B, and C. Each bookshelf requires a certain amount of time for cutting each component, assembling them, and then staining them. WBF also sells unstained model C bookshelves. The times, in hours, for each phase of construction and the profit margins for each model, as well as the amount of time available in each department are given below.

Model	Cutting	Assembling	Staining	Profit Margin
A	3	4	5	\$20
B	1	2	5	\$25
C	4	5	4	\$50
Unstained C	4	5	0	\$30
Labor available	150	185	250	

During the holiday season, WBF can sell every unit that it makes. How many of each model should be produced if WBF wants to maximize its holiday profits?

If we let A , B , C , and UC denote the number of models A, B, C, and unstained C bookshelves produced, respectively, it is easy to see that the

linear program derived to solve this problem is

$$\begin{aligned}
 \max \quad & 20A + 25B + 50C + 30UC \\
 \text{s.t.} \quad & \\
 & 3A + 1B + 4C + 4UC \leq 150 \quad (\text{Cutting}) \\
 & 4A + 2B + 5C + 5UC \leq 185 \quad (\text{Assembling}) \\
 & 5A + 5B + 4C \leq 250 \quad (\text{Staining}) \\
 & A, B, C, UC \geq 0.
 \end{aligned}$$

The optimal solution is to produce 30 model B and 25 (stained) model C bookshelves, generating a profit of \$2000; thus, we do not need to indicate that the variables are integer-valued in our model.

Now, suppose that we want to buy out WBF by purchasing all its resources. How much should be offered for each hour of cutting, assembling, and staining? For example, if we offered WBF \$3 for each hour in every department, would WBF accept it? Well, suppose WBF were to make one unit of model C. Its profit margin is \$50. It requires 4 hours of cutting, 5 hours of assembling, and 4 hours of staining. Under our proposal, which would generate only $\$3(4 + 5 + 4) = \39 , implying that it is better for them to produce a unit of model C than to accept this offer.

After trying this approach, we reason that if we were to offer them $\$y_1$ for each hour in the cutting department, $\$y_2$ for each hour in the assembling department, and $\$y_3$ in the staining department, we would need to make sure that it is not more beneficial for them to produce a unit of any model than it is to accept our offer. To arrange this, we rationalize that our offer must satisfy the following conditions:

$$\begin{aligned}
 3y_1 + 4y_2 + 5y_3 &\geq 20 \\
 y_1 + 2y_2 + 5y_3 &\geq 25 \\
 4y_1 + 5y_2 + 4y_3 &\geq 50 \\
 4y_1 + 5y_2 &\geq 30 \\
 y_1, y_2, y_3 &\geq 0.
 \end{aligned}$$

Since we obviously want to spend as little money as possible, and the total amount spent under this plan is $150y_1 + 185y_2 + 280y_3$, the best plan could be found as the solution to the linear program

$$\begin{aligned}
 \min \quad & 150y_1 + 185y_2 + 280y_3 \\
 \text{s.t.} \quad &
 \end{aligned}$$

$$\begin{aligned}
3y_1 + 4y_2 + 5y_3 &\geq 20 \\
y_1 + 2y_2 + 5y_3 &\geq 25 \\
4y_1 + 5y_2 + 4y_3 &\geq 50 \\
4y_1 + 5y_2 &\geq 30 \\
y_1, y_2, y_3 &\geq 0.
\end{aligned}$$

Note that this is exactly the dual program to our original linear program! The optimal solution to the primal problem is $(A, B, C, UC) = (0, 30, 25, 0)$ and $(y_1, y_2, y_3) = (0, \frac{150}{17}, \frac{25}{17})$ is the optimal dual solution. Each optimal value is 2000, complementary slackness conditions are met, and each solution is nondegenerate.

We initially formed the dual problem to determine the worth of the company if we were interested in purchasing all its resources. We would need to spend at least \$2000 to entice WBF to sell their business. However, we can use these dual variable values in another way. Suppose WBF were interested in adding additional resources (in this case, labor hours) so as to increase production. Where should they spend the money? Note that the optimality of both the primal and the dual solutions generates

$$2000 = 150y_1 + 185y_2 + 250y_3.$$

Since they do not use all cutting hours (there is an excess of 20 hours available), it is not worth any money to purchase additional hours. This is reflected in the fact that the dual value $y_1 = 0$ for this constraint; indeed, increasing the available cutting hours from 150 does not alter the optimal value. However, since $y_2 = \frac{150}{17} = 8.8235$, increasing the number of hours in the assembling department from 185 would increase the optimal profit by roughly \$8.82 per hour. Similarly, since $y_3 = \frac{25}{17} = 1.4706$, increasing the hours in staining would increase the optimal profit by roughly \$1.47 per hour. This **marginal cost** or **shadow price** of a resource can be seen to be related to the value of its dual variable. It should be easy to see that this shadow price is not valid for every change in the resource value. For example, if they were to increase the number of hours in the assembling department by 1000 hours, their profit would not increase by $1000y_2$ since other constraints would become active in this revised problem.

We will explore when these shadow prices are valid in Chapter 10. In addition, these shadow prices can be affected by the presence of degeneracy

in our primal problem and so are not always equivalent to the value of the constraint's dual variable (see Exercise 9.19).

9.7 ANOTHER DUALITY APPROACH: LAGRANGIAN DUALITY

Now that we've seen linear programming duality, studied some of its properties, and seen what new perspectives it brings to linear programming, let's turn our attention to another duality approach. As we mentioned at the beginning of this chapter, one of the main motivations for constructing a dual problem to our primal problem is to generate an appropriate bound to the primal value. In fact, any problem that routinely satisfies such a weak duality result can be justifiably called a dual problem.

For general mathematical programs, and especially for integer programs, there are many other classes of dual programs available for use. One of the earliest, and perhaps most used, approaches is related to the concept of Lagrange multipliers that is studied in calculus classes.

To illustrate this approach, consider the following linear program:

$$\begin{aligned}
 \max \quad & \sum_{j=1}^n c_j x_j \\
 \text{s.t.} \quad & \sum_{j=1}^n a_{1j} x_j \leq b_1 \\
 & \sum_{j=1}^n a_{2j} x_j \leq b_2 \\
 (9.16) \quad & x_j \geq 0, \quad j \in \{1, \dots, n\}.
 \end{aligned}$$

Our goal is to construct a dual problem that satisfies a weak duality result, which means that every feasible solution to (9.16) will have objective value no larger than a feasible solution to the dual. To this end, suppose we remove the constraints from (9.16) and place them in the objective function, after multiplying it by some multipliers (variables) λ_1, λ_2 ; this generates the

problem, for each value of λ_1, λ_2 ,

$$(9.17) \quad \begin{aligned} g(\lambda_1, \lambda_2) &= \max \sum_{j=1}^n c_j x_j + \lambda_1 \left(b_1 - \sum_{j=1}^n a_{1j} x_j \right) + \lambda_2 \left(b_2 - \sum_{j=1}^n a_{2j} x_j \right) \\ \text{s.t.} \quad x_j &\geq 0. \end{aligned}$$

Note that every feasible solution \mathbf{x} to (9.16) is also feasible to (9.17), which means

$$b_1 - \sum_{j=1}^n a_{1j} x_j \geq 0 \quad \text{and} \quad b_2 - \sum_{j=1}^n a_{2j} x_j \geq 0,$$

and so we get weak duality as long as $\lambda_1, \lambda_2 \geq 0$. Again, we want to have our upper bound $g(\lambda_1, \lambda_2)$ as small as possible, giving us the dual problem

$$(9.18) \quad \begin{aligned} z_D &= \min_{\lambda_1, \lambda_2 \geq 0} g(\lambda_1, \lambda_2) \\ &= \min_{\lambda_1, \lambda_2 \geq 0} \max_{x_j \geq 0} \sum_{j=1}^n c_j x_j + \lambda_1 \left(b_1 - \sum_{j=1}^n a_{1j} x_j \right) \\ &\quad + \lambda_2 \left(b_2 - \sum_{j=1}^n a_{2j} x_j \right). \end{aligned}$$

Similar thinking would lead us to the dual problem

$$(9.19) \quad \begin{aligned} z_{D_2} &= \min_{\lambda_1 \geq 0} h(\lambda_1) \\ &= \min_{\lambda_1 \geq 0} \max_{x_j \geq 0} \left\{ \sum_{j=1}^n c_j x_j + \lambda_1 \left(b_1 - \sum_{j=1}^n a_{1j} x_j \right) : \sum_{j=1}^n a_{2j} x_j \leq b_2 \right\}, \end{aligned}$$

where we placed only the first constraint into the objective. Note that $h(\lambda_1)$ also satisfies weak duality.

Consider the primal mathematical program

$$(9.20) \quad \begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in P, \end{aligned}$$

where P is any set of constraints. These could include, but are not limited to, $\mathbf{x} \geq \mathbf{0}$, $D\mathbf{x} = \mathbf{d}$, or even x_j is an integer. We associate a vector of multipliers $\lambda \geq \mathbf{0}$ with the system of constraints $A\mathbf{x} \leq \mathbf{b}$ and generate the

dual problem

$$(9.21) \quad z_{LD} = \min_{\lambda \geq 0} \max_{\mathbf{x} \in P} \mathbf{c}^T \mathbf{x} + \lambda^T (\mathbf{b} - A\mathbf{x}).$$

This problem is known as the **Lagrangian dual** of our primal problem, where each λ_j is known as a **Lagrange multiplier**.

It is easy to see that the Lagrangian dual problem (9.21) generates a weak duality result (see Exercise 9.20). Unfortunately, for many mathematical programs, we cannot guarantee a strong duality result. In fact, linear programs are one of the few that actually have a dual problem satisfying strong duality, which makes linear programming duality so powerful.

Let's now return to our original example (9.16) and consider again the function $g(\lambda_1, \lambda_2)$ in (9.17). We can rewrite this as

$$\begin{aligned} g(\lambda_1, \lambda_2) &= \max_{\mathbf{x} \geq 0} \quad \sum_{j=1}^n c_j x_j + \lambda_1 \left(b_1 - \sum_{j=1}^n a_{1j} x_j \right) + \lambda_2 \left(b_2 - \sum_{j=1}^n a_{2j} x_j \right) \\ &= \max \quad \lambda_1 b_1 + \lambda_2 b_2 + \sum_{j=1}^n (c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2) x_j \\ &\text{s.t.} \\ &\quad x_j \geq 0. \end{aligned}$$

If we choose $\lambda_1, \lambda_2 \geq 0$ so that $c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2 > 0$, then $g(\lambda_1, \lambda_2) \rightarrow \infty$, since we can make each x_j arbitrarily large. Hence, to ensure that $g(\lambda_1, \lambda_2)$ has a finite value, we need to further restrict those values of $\lambda_1, \lambda_2 \geq 0$ to

$$(9.22) \quad c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2 \leq 0, \quad j \in \{1, \dots, n\}.$$

Does this restriction/constraint look familiar? It should—it would be the constraint of the dual linear program to (9.16)! Looking further at the Lagrangian dual, we see that, under (9.22), we can further generate the value of $g(\lambda_1, \lambda_2)$ by noting that if $c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2 < 0$, then we must have $x_j = 0$. This implies that

$$g(\lambda_1, \lambda_2) = \lambda_1 b_1 + \lambda_2 b_2$$

under the conditions

$$\begin{aligned}
c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2 &\leq 0, & j \in \{1, \dots, n\} \\
(c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2)x_j &= 0, & j \in \{1, \dots, n\} \\
x_j &\geq 0, & j \in \{1, \dots, n\} \\
\lambda_1, \lambda_2 &\geq 0.
\end{aligned}$$

Note that the second condition is, in fact, complementary slackness! Thus, our Lagrangian dual problem would be

$$\begin{aligned}
\min \quad & \lambda_1 b_1 + \lambda_2 b_2 \\
\text{s.t.} \quad & c_j - a_{1j}\lambda_1 - a_{2j}\lambda_2 \leq 0, \quad j \in \{1, \dots, n\} \\
(9.23) \quad & \lambda_1, \lambda_2 \geq 0,
\end{aligned}$$

which is the dual to (9.16). We already had weak duality, we then derived complementary slackness conditions, and now we know that strong duality holds here.

This example highlights an important note regarding Lagrangian duals of linear programs.

Given a linear program, its Lagrangian dual is equal to its dual linear program. The Lagrange multipliers λ are the variables to the dual linear program.

Lagrangian duality is more general than linear programming duality since (a) we do not need to “dualize” every constraint and (b) it can handle nonlinear constraints as well, including integer variables. We will return to this notion in Chapter 13 when we explore integer programs.

Note on KKT Conditions We now provide a quick introduction to the use of Lagrangian duals for nonlinear programs;¹ for a more detailed introduction, consider the books by Bazaar et al. [9] and Nash and Sofer [66]. Consider the following nonlinear program:

$$\begin{aligned}
\max \quad & f(\mathbf{x}) \\
\text{s.t.} \quad & g(\mathbf{x}) = b,
\end{aligned}$$

where $f(\mathbf{x})$ and $g(\mathbf{x})$ are differentiable functions of \mathbf{x} . If we take a similar approach to above, our Lagrangian function would be

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda(b - g(\mathbf{x})),$$

where λ is unrestricted in sign due to the equality constraint. Recalling the Lagrange multiplier approach from calculus, a necessary condition for \mathbf{x} and λ to be optimal is

$$\begin{aligned} \nabla f(\mathbf{x}) - \lambda \nabla g(\mathbf{x}) &= 0 \\ (9.24) \quad g(\mathbf{x}) &= b. \end{aligned}$$

This can be easily extended to m constraints $g_k(\mathbf{x}) = b_k$, where the necessary conditions (9.24) become

$$\begin{aligned} \nabla f(\mathbf{x}) - \sum_{k=1}^m \lambda_k \nabla g_k(\mathbf{x}) &= 0 \\ (9.25) \quad g_k(\mathbf{x}) &= b_k, \quad k \in \{1, \dots, m\}, \end{aligned}$$

provided there exist n gradients ∇g_k that are linearly independent at the solution \mathbf{x}^* to (9.25).

Now, suppose we have an optimization problem with inequality constraints

$$\min f(\mathbf{x})$$

s.t.

$$(9.26) \quad g_k(\mathbf{x}) \geq b_k, \quad k \in \{1, \dots, m\},$$

where f and each g_k is differentiable in \mathbf{x} ; note that these constraints could include $x_j \geq 0$. As we did for linear programs, to each constraint we assign a variable $\lambda_k \geq 0$ and form the Lagrangian function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_{k=1}^m \lambda_k (b_k - g_k(\mathbf{x})).$$

At an optimal solution \mathbf{x}^* , those constraints k that are not active contribute to the value of $L(\mathbf{x}, \lambda)$ if $\lambda_k > 0$, which implies that we would want $\lambda_k = 0$ for inactive constraints; this condition can be rewritten as

$$\lambda_k (b_k - g_k(\mathbf{x})) = 0,$$

which are complementary slackness conditions. In fact, the necessary conditions for \mathbf{x} to be optimal are known as the Karush–Kuhn–Tucker (KKT) conditions:

$$\begin{aligned}
\nabla f(\mathbf{x}) - \sum_{k=1}^m \lambda_k \nabla g_k(\mathbf{x}) &= 0 \\
\lambda_k (b_k - g_k(\mathbf{x})) &= 0 \\
g_k(\mathbf{x}) &\geq b_k, \quad k \in \{1, \dots, m\} \\
(9.27) \quad \lambda_k &\geq 0.
\end{aligned}$$

Note that these conditions generalize the primal feasibility, dual feasibility, and complementary slackness conditions of linear programs. To make these necessary conditions, some qualifications on the constraints are needed. One simple condition is that the gradients of the functions g_k active at \mathbf{x}^* are linearly independent. For example, if the constraints are $A\mathbf{x} \geq \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$, we would need A to have rank of m .

Much like linear programming duality helps guide the simplex method toward an optimal solution, KKT conditions guide algorithms designed to solve nonlinear programs. Unfortunately, typically solutions that satisfy the KKT conditions are only local optimal solutions. However, if $f(\mathbf{x})$ is a convex function and the feasible region is a convex set, a solution satisfying the KKT conditions is globally optimal.

Summary

In this chapter, we introduced the notion of duality to linear programming and examined its consequences. This included the construction of a related dual linear program that provides a bound to our primal problem and whose optimal value equals that of our primal. We further note that the values of the primal (dual) variables are complementary to the slack (surplus) of the dual (primal) constraints. It is these strong duality and complementary slackness conditions that form a certificate of optimality for our primal solution. In addition, we saw that the values of the dual variables have an economic interpretation, and that there are additional dual problems that we can formulate, such as the Lagrangian dual.

However, this was just an introduction. Duality plays such a fundamental role in linear programming that it is hard to overemphasize. As mentioned above, the simplex method, when looked at through the lens of duality theory, is an algorithm that maintains primal feasibility and complementary slackness, and systematically finds a dual feasible solution, if one exists. Such a perspective allows us to develop a much richer theory of linear

programming, including both novel applications and different algorithmic approaches, which we explore in the next two chapters.

EXERCISES

State the dual of each linear program in Exercises 9.1–9.4.

9.1

$$\max \quad 8x_1 + 20x_2 + 4x_3$$

s.t.

$$2x_1 + 3x_2 + 2x_3 \leq 20$$

$$x_2 + x_3 \leq 8$$

$$3x_1 + x_2 - 4x_3 \geq 10$$

$$x_1, x_2, x_3 \geq 0.$$

9.2

$$\max \quad 30x_1 - 2x_3 + 10x_4$$

s.t.

$$2x_1 - 3x_2 + 9x_4 \leq 10$$

$$4x_2 - x_3 \geq 19$$

$$x_1 + x_2 + x_3 = 5$$

$$x_1 \geq 0, x_3 \leq 0$$

9.3

$$\min \quad 5x_1 + x_2 - 4x_3$$

s.t.

$$x_1 + x_2 + x_3 + x_4 = 19$$

$$4x_2 + 8x_4 \leq 55$$

$$x_1 + 6x_2 - x_3 \geq 7$$

$$x_2, x_3 \geq 0, x_4 \leq 0$$

9.4

$$\begin{aligned}
\min \quad & 2x_1 - 7x_2 + 6x_3 + 5x_4 \\
\text{s.t.} \quad & \\
& 2x_1 - 3x_2 - 5x_3 - 4x_4 \leq 20 \\
& 7x_1 + 2x_2 + 6x_3 - 2x_4 = 35 \\
& 4x_1 + 5x_2 - 3x_3 - 2x_4 \geq 15 \\
& 0 \leq x_1 \leq 10 \\
& 0 \leq x_2 \leq 5 \\
& x_3 \geq 2 \\
& x_4 \geq 0
\end{aligned}$$

9.5 Consider the linear program

$$\begin{aligned}
\max \quad & 5x + y \\
\text{s.t.} \quad & \\
& x + y \leq 7 \\
& x - y \leq 3 \\
& x, y \geq 0.
\end{aligned}$$

(a) Derive the dual linear program.

(b) Plot the feasible regions to both the primal and the dual problems.

(c) Solve the primal problem using the simplex method. In each iteration, indicate on the graphs both the current primal basic feasible solution and its corresponding dual solution.

9.6 Consider the following linear program:

$$\begin{aligned}
\max \quad & 9x_1 + 14x_2 + 7x_3 \\
\text{s.t.} \quad & \\
& 2x_1 + x_2 + 3x_3 \leq 6 \\
& 5x_1 + 4x_2 + x_3 \leq 12 \\
& 2x_2 \leq 5 \\
& x_1, x_2, x_3 \geq 0.
\end{aligned}$$

(a) Show using the reduced costs that $\mathbf{x} = (\frac{5}{26}, \frac{5}{2}, \frac{27}{26})$ is an optimal solution. Do not go through the entire simplex method!

(b) What is the dual of this linear program?

(c) Find the optimal solution to the dual using the optimal solution given in (a).

9.7 Suppose that $x_1 = 2, x_2 = 0, x_3 = 4$ is an optimal solution to the linear

program

$$\max \quad 4x_1 + 2x_2 + 3x_3$$

s.t.

$$2x_1 + 3x_2 + x_3 \leq 12$$

$$x_1 + 4x_2 + 2x_3 \leq 10$$

$$3x_1 + x_2 + x_3 \leq 10$$

$$x_1, x_2, x_3 \geq 0.$$

Using only complementary slackness and the strong duality theorem, find an optimal solution to the dual problem.

9.8 Suppose that $y_1 = 0, y_2 = 1, y_3 = 2$ is an optimal solution to the dual of the linear program

$$\max \quad 2x_1 + x_2$$

s.t.

$$x_1 - 2x_2 \leq 2$$

$$2x_1 - x_2 \leq 7$$

$$x_2 \leq 3$$

$$x_1, x_2 \geq 0.$$

Using only complementary slackness and the strong duality theorem, find an optimal solution to the given primal problem.

9.9 Consider the linear program

$$\max \quad 5x - 4y$$

s.t.

$$-3x + 2y \leq -1$$

$$x - y \leq 1$$

$$-2x + 7y \leq 6$$

$$9x - 7y \leq 10$$

$$-5x + 2y \leq -3$$

$$7x - 3y \leq 11$$

$$x, y \geq 0.$$

(a) Solve this problem using the two-phase method given in Chapter 8, using the simplex method in each phase.

(b) State the dual linear program.

(c) Find the optimal solution to the linear program by solving its dual. Was this easier than solving the problem in (a)? Explain.

9.10 Suppose the following two constraints are part of some linear program. What is the value of the dual variable corresponding to the second constraint? Explain. Assume all variables are nonnegative.

$$2x_1 + 5x_2 + 3x_3 \leq 10$$

$$4x_1 + 9x_2 + 6x_3 \leq 21.$$

9.11 Prove the strong duality theorem (Theorem 9.2) using Farkas' lemma.

Hint: Mimic the argument presented in Section 9.5.

9.12 Prove the following version of Farkas' lemma using linear programming duality—one and only one of the following two systems has a solution:

$$\text{System 1: } Ax \leq \mathbf{0} \text{ and } c^T x > 0$$

$$\text{System 2: } A^T y = c \text{ and } y \geq \mathbf{0}.$$

9.13 We saw in Sections 6.4 and 9.5 how Farkas' lemma can be interpreted as determining when an improving feasible direction exists for an LP in standard form. Use the version of Farkas' lemma in Exercise 9.12 above to find the version for an LP in canonical form. *Hint:* System 1 will be

$$Ad = \mathbf{0}, d_N \geq \mathbf{0} \text{ and } c^T d > 0.$$

9.14 Prove the second part of the complementary slackness theorem, namely, that $x_j^* w_j^* = 0$ for $j = 1, 2, \dots, n$.

9.15 Consider the following linear programming problems:

$$\begin{aligned} & \max \quad c^T x \\ & \text{s.t.} \\ & \quad Ax \leq b. \end{aligned}$$

and

$$\begin{aligned} & \min \quad c^T x \\ & \text{s.t.} \\ & \quad Ax \geq b \end{aligned}$$

(a) If both problems are feasible, prove that if one of these problems has a finite optimal solution, so does the other.

(b) Assume both have finite optimal solutions. Let \mathbf{x}_1 be optimal to the first

(maximization) problem and \mathbf{x}_2 be optimal to the second (minimization) problem. What is the relationship between $\mathbf{c}^T \mathbf{x}_1$ and $\mathbf{c}_2^T \mathbf{x}_2$? Prove your claim.

9.16 Prove that if the problem

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

has a finite optimal solution, then the new problem

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \hat{\mathbf{b}}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

cannot be unbounded for any choice of the vector $\hat{\mathbf{b}}$.

9.17 Suppose, we are given the linear program

$$\begin{aligned} & \max \quad c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{s.t.} \quad & a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b \\ & x_i \leq 1, \quad i \in \{1, 2, \dots, n\} \\ & x_i \geq 0, \end{aligned}$$

where the variables are ordered such that

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}.$$

Let r be the index such that

$$\sum_{i=1}^{r-1} a_i \leq b \text{ and } \sum_{i=1}^r a_i > b.$$

Use complementary slackness and the strong duality theorem to show that the solution

$$x_k = \begin{cases} 1, & \text{if } k < r \\ \frac{b - \sum_{i=1}^{r-1} a_i}{a_r}, & \text{if } k = r \\ 0, & \text{if } k > r \end{cases}$$

is optimal. For example, consider the LP

$$\begin{aligned}
\max \quad & 8x_1 + 5x_2 + 9x_3 + 10x_4 + 5x_5 \\
\text{s.t.} \quad & \\
& 2x_1 + 2x_2 + 4x_3 + 5x_4 + 3x_5 \leq 12 \\
& x_1, x_2, x_3, x_4, x_5 \leq 1 \\
& x_1, x_2, x_3, x_4, x_5 \geq 0
\end{aligned}$$

has as an optimal solution the point $(1, 1, 1, \frac{4}{3}, 0)$. Hint: Each $x_i \leq 1$ is an individual constraint that you must account for.

9.18 Consider the 0–1 knapsack problem

$$z_{IP} = \max \quad \sum_{j=1}^n c_j x_j$$

s.t.

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

$$x_j \in \{0, 1\},$$

$$0 \leq b \leq \sum_{j=1}^n a_j.$$

where each $c_j, a_j \geq 0$ and Assume that

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \dots \geq \frac{c_n}{a_n}.$$

Its Lagrangian dual is

$$z_{LD} = \min_{\lambda \geq 0} \max_{\mathbf{x} \in \{0, 1\}^n} \sum_{j=1}^n c_j x_j + \lambda \left(b - \sum_{j=1}^n a_j x_j \right).$$

(a) Show that this dual problem satisfies weak duality. That is, show that $z_{IP} \leq z_{LD}$.

(b) Prove that $z_{LD} \leq z_{LP}$, where z_{LP} is the value of the LP-relaxation to the problem.

9.19 Consider the following primal problem:

$$\max \quad 3x + 8y + z$$

s.t.

$$x + 4y + 5z \leq 20$$

$$x + y + 3z \leq 8$$

$$2x + 3y + z \leq 20$$

$$x + 2y + 2z \leq 12$$

$$x, y, z \geq 0.$$

An optimal solution is $(x, y, z) = (4, 4, 0)$ and the corresponding dual optimal solution is $(y_1, y_2, y_3, y_4) = (1.4, 0, 0.8, 0)$. Note that the optimal basic feasible solution to the primal problem is degenerate. Show that decreasing the righthand side of constraint 4 by 1 decreases the value of the optimal solution even though the corresponding dual variable value is 0.

9.20 Show that the Lagrangian dual problem (9.21) satisfies weak duality for its primal problem (9.20).

9.21 In Section 2.9, we saw the transportation problem, which was stated as

$$\begin{aligned} \min \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^n x_{ij} = s_i, \quad i \in \{1, \dots, m\} \\ & \sum_{i=1}^m x_{ij} = d_j, \quad j \in \{1, \dots, n\} \\ & x_{ij} \geq 0, \quad i \in \{1, \dots, m\}, \quad j \in \{1, \dots, n\} \end{aligned}$$

where

$$\sum_{i=1}^m s_i = \sum_{j=1}^n d_j.$$

State the dual of this linear program by letting u_i be the dual variables corresponding to the first m constraints and v_j the dual variables corresponding to the remaining constraints.

9.22 In Exercise 2.43, we saw that the shortest path problem can be formulated as the linear program

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = \begin{cases} n-1, & i=s, \\ -1, & i \in V - \{s\}, \end{cases} \\ & x_{ij} \geq 0, \quad (i, j) \in A. \end{aligned}$$

Find the dual to this problem. (*Hint:* The dual variables y_i correspond to the

nodes $i \in V$ of the network and the dual constraints correspond to the arcs $(i, j) \in A.$)

9.23 Given the linear program

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{Ax} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \end{aligned}$$

where \mathbf{l}, \mathbf{u} satisfy $\mathbf{l} \leq \mathbf{u},$

- (a)** Derive the dual to the linear program.
- (b)** Prove that the dual always contains a feasible solution.
- (c)** What does this imply about the primal problem?

¹This is a very brief introduction to the topic and is meant to simply highlight the “bigger picture.” It may be omitted without loss of continuity

CHAPTER 10

SENSITIVITY ANALYSIS OF LINEAR PROGRAMS

In many applications, the values of a linear program's parameters may change or are merely estimates of their actual (unknown) value. When a single parameter does change, *Sensitivity Analysis* often makes it unnecessary to resolve the problem again in order to obtain the new optimal value to the objective function. This can be of great use, especially when our linear program is very large and resolving it can take a long time. It is important to note that **sensitivity analysis is based on the proposition that all data except for one number in the model are held fixed**. In this chapter, we first look at sensitivity analysis from a graphical point of view for two-variable linear programs. Next, we use our knowledge of the simplex method and linear programming duality to derive these values for general linear programs and explore examples that illustrate how to interpret sensitivity analysis results. Finally, we examine the effect of changing multiple objective coefficients or right-hand-side values.

10.1 GRAPHICAL SENSITIVITY ANALYSIS

When we model real-world situations, we are often not 100% sure if the data we're using are correct. In these cases, it is often useful to determine what effect the change of one bit of data has on our model. In this section, we explore the effect on the optimal solution from various changes to a two-variable linear programming model: adding or removing constraints, changing the right-hand side of a single constraint, and changing the

objective function coefficient of a single variable. We use the fact that two-variable linear programs can be easily viewed in a graphical sense. Whenever we do any of these changes, we always assume that **the remaining data are unchanged**.

In this section, we consider the linear program

$$(10.1a) \max z = 13x + 5y$$

s.t.

$$(10.1b) 4x + y \leq 24$$

$$(10.1c) x + 3y \leq 24$$

$$(10.1d) 3x + 2y \leq 23$$

$$(10.1e) x, y \geq 0.$$

Its feasible region is given in [Figure 10.1](#) and examining each of its extreme points

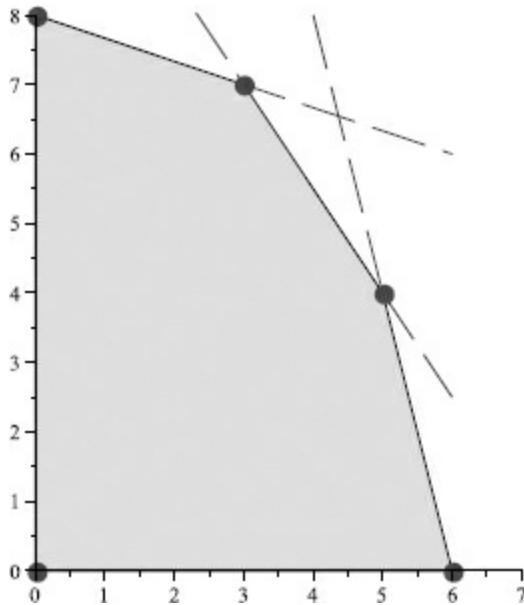
$$V = \{(0, 0), (0, 8), (3, 7), (5, 4), (6, 0)\},$$

we find that its optimal solution is (5, 4). Throughout this section, we examine what effect changes to this linear program has on its optimal solution.

Adding or Removing Constraints

One change that is often made to linear programs during the modeling phase is the addition or removal of constraints. This is because the modeler either identified new constraints that were not apparent the first time or decided that certain constraints were not necessary. It is therefore useful to determine what effect these changes have on the optimal solution to a linear program.

[FIGURE 10.1](#) Feasible region for linear program (10.1).



Removing Constraints Suppose that we remove the constraint

$$x + 3y \leq 24$$

from (10.1). How does the optimal objective function value change? First, notice that every solution that was feasible in (10.1) is still feasible, including (5, 4). In fact, resolving the problem without this constraint reveals that the optimal solution is still (5, 4). What happens if we remove the constraint

$$4x + y \leq 24?$$

In this case, resolving the problem yields an optimal solution of $(7\frac{2}{3}, 0)$ with value $\frac{99}{3}$. Notice that the point (5, 4) is feasible in this updated problem, but is no longer optimal.

Does this happen when any constraint is removed? Yes. In fact, what we are doing is **relaxing** the feasible region, that is, enlarging the feasible region. In this case, our original feasible region is a subset of the new feasible region. When this happens, since the “old” optimal solution is still feasible, our “new” optimal value cannot be worse than the “old” value.

We’ve actually relaxed some feasible regions before, as when in Section 5.4 we made the integer variables continuous in a linear programming relaxation. This also occurs when we increase the right-hand side of a “ \leq ” constraint or decrease the right-hand side of a “ \geq ” constraint. In each of these cases, our new optimal value would not be worse than our old value.

Given a mathematical program

$$z^* = \max \{f(\mathbf{x}) : \mathbf{x} \in S\},$$

let $S \subseteq S_0$ (i.e., relax the feasible region). The following inequality holds:

$$\max \{f(\mathbf{x}) : \mathbf{x} \in S_0\} \geq z^*.$$

Hence, in **any** mathematical program, relaxing the feasible region cannot make the optimal value of a mathematical program worse (the optimal value of a maximized (minimized) objective function cannot decrease (increase)).

Adding Constraints What about adding an additional constraint? For example, suppose we add the constraint $x + y \leq 8$ to (10.1). The previously optimal solution $(5, 4)$ is no longer feasible. Hence, our optimal value cannot improve (i.e., increase). Of course, if we had added the constraint $x + y \leq 9$, no change in the optimal value would have occurred, since our previous optimal solution would remain feasible.

What we have done is **tighten** the feasible region, or eliminate previously feasible solutions. This occurs when we add constraints, reduce the right-hand side of “ \leq ” constraints, increase the right-hand side of “ \geq ” constraints, or make continuous variables into discrete variables. In each case, the new feasible region is contained entirely within the old feasible region. Thus, the optimal value over the new feasible region cannot be better than the optimal value over the old feasible region (since the old optimal solution may no longer be feasible).

Given a mathematical program

$$z^* = \max \{f(\mathbf{x}) : \mathbf{x} \in S\},$$

let $S_0 \subseteq S$ (i.e., tighten the feasible region). Then

$$\max \{f(\mathbf{x}) : \mathbf{x} \in S_0\} \leq z^*.$$

Hence, in **any** mathematical program, tightening the feasible region cannot make the optimal value of a mathematical program better (the optimal value of a maximized (minimized) objective function cannot increase (decrease)).

A special case occurs when we add a constraint and no change occurs to the feasible region; that is, every point in the old feasible region satisfies this constraint.

Redundant Constraint A constraint is *redundant* if its removal from or addition to the list of constraints to a mathematical program does not alter the feasible region.

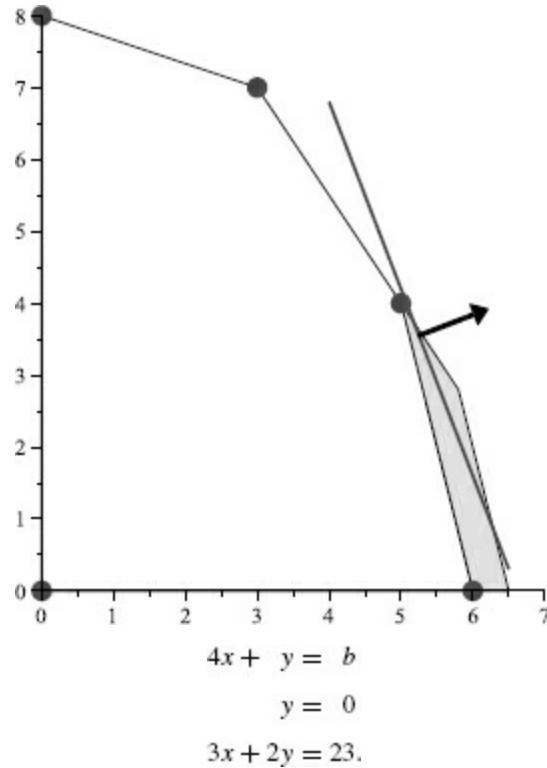
It should be obvious that any redundant constraint can be removed from the mathematical program. However, this can be done only one constraint at a time, at which point all previously redundant constraints should be re-evaluated to see if they are still redundant (see Exercise 10.7).

Change in Right-Hand Side

Now consider what happens when we change some of the parameters of a linear program. Suppose we began with the linear program (10.1) and we found that the right-hand side of (10.1b) should have been 26 instead of 24; that is, the constraint should be $4x + y \leq 26$. [Figure 10.2](#) shows both the old feasible region and the new one, where the additional region is shaded. Is (5, 4) still the optimal solution with value 85? No, because the contour line $13x + 5y = 86$ intersects the feasible region (see [Figure 10.2](#)). However, the optimal solution is still at the intersection of the first and third constraints. So the real question that should be asked is, by how much can we increase or decrease the right-hand side of a constraint and still have the optimal solution at the intersection of the same constraints? At the same time, we can ask by how much does the optimal value change with each unit change in the right-hand side.

Originally, the optimal solution occurs at the intersection of the constraints (10.1b) and (10.1d). Notice how, as we increase the right-hand side of (10.1b), the extreme point “slides down” toward the x -axis. Remember that the x -axis is itself a constraint (because of (10.1e)), and so it appears that we can increase the right-hand side of (10.1b) until the “optimal extreme point” intersects with another constraint. This is because, if we increase the right-hand side of (10.1b) any more, the optimal solution will occur at the intersections of $y = 0$ and (10.1d). In this case, we want to find the value of b that solves

FIGURE 10.2 Change in right-hand side.



This occurs at the point $(\frac{23}{3}, 0)$, giving the value $b = 4\left(\frac{23}{3}\right) = 30\frac{2}{3}$. Hence, we can increase the right-hand side of (10.1b) to at most $30\frac{2}{3}$ and know that our optimal solution will still occur at the intersection of constraints (10.1b) and (10.1d).

If we decrease the right-hand side of (10.1b), then the extreme point slides toward the point $(3, 7)$, or the intersection of the constraints (10.1c) and (10.1d). Hence, we would need to find the value of b that solves

$$\begin{aligned} 4x + y &= b \\ x + 3y &= 24 \\ 3x + 2y &= 23. \end{aligned}$$

Since $(x, y) = (3, 7)$ solves this system, $b = 4(3) + 7 = 19$, and as long as the right-hand side b of (10.1b) satisfies $19 \leq b \leq 30\frac{2}{3}$, the optimal solution to (10.1) occurs at the intersection of constraints (10.1b) and (10.1d).

Now consider how the optimal value of (10.1) changes when the right-hand side of (10.1b) changes. Suppose that the right-hand side of (10.1b) changes by Δ to $24 + \Delta$. Since the optimal solution occurs at the intersection of the lines

$$(10.2) \quad 4x + y = 24 + \Delta$$

$$(10.3) \quad 3x + 2y = 23,$$

Whenever $-5 \leq \Delta \leq 6\frac{2}{3}$ (see above), the optimal solution is $(5 + \frac{2}{5}\Delta, 4 - \frac{3}{5}\Delta)$ (check this!), and its optimal value is

$$13(5 + \frac{2}{5}\Delta) + 5(4 - \frac{3}{5}\Delta) = 85 + \frac{11}{5}\Delta.$$

Thus, every unit change of Δ of within the range $-5 \leq \Delta \leq 6\frac{2}{3}$ yields a change in the optimal value by $\frac{11}{5}$. This rate of change of the optimal value is the *shadow price* of the constraint, which we first saw in Section 9.6. Note that if Δ is not within this range, then this rate of change is no longer valid because the optimal solution would not be at the intersection of the lines (10.2) and (10.3).

Shadow Price The *shadow price* of a constraint is the amount that the optimal value of a linear program changes with per unit change in the right-hand side of the constraint (assuming that the current basis remains optimal).

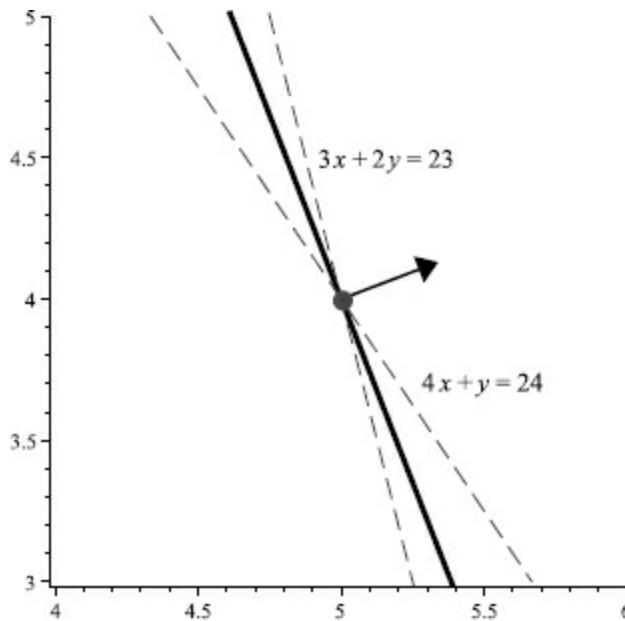
We can do the same analysis for the other constraints (10.1c) and (10.1d). Similar calculations find that the right-hand side $b = 23$ of constraint (10.1c) can be adjusted by $-5 \leq \Delta \leq \frac{35}{11}$, with a shadow price of 1.4. For constraint (10.1d), the situation is more interesting. Here, the constraint does not go through the optimal solution, so any change in the right-hand side of this constraint has no impact on the optimal value of the linear program; thus, its shadow price would be 0. For what values of the right-hand side is this valid? Again, the solution $(5, 4)$ would remain optimal until $(5, 4)$ is on the constraint $x + 3y = 24 + \Delta$. This occurs when $\Delta = -7$. Now, how about increasing the right-hand side? As Δ increases, our feasible region is relaxed, and so our optimal value can increase. However, checking [Figure 10.2](#) indicates that these added solutions have lower objective function value than the solution $(5, 4)$. In fact, if $\Delta \geq 10.5$ (or $b \geq 34.5$), we get the constraint

$$x + 3y \leq 34.5,$$

and this constraint becomes redundant (check this!). Hence, we can increase the right-hand side of (10.1d) indefinitely and not change the optimal solution, implying our optimal solution remains optimal as long as the right-

hand side remains in the range $17 \leq b < \infty$.

FIGURE 10.3 Change in objective function coefficient.



Change in the Objective Function

What effect does changing the coefficient of x from 13 to 12 have on the optimal solution? [Figure 10.3](#) shows the contour for $13x + 5y = 85$ along with the constraint lines $3x + 2y = 23$ and $4x + y = 24$. Note that, as the coefficient of x changes, the slope of the objective function line changes.

We know that if the objective function “matches,” say, the constraint $3x + 2y = 23$, we would have multiple optimal solutions. The objective function contour would then be $7.5x + 5y = 57.5$ (remember, we want the coefficient of y to remain 5). If the coefficient of x drops below 7.5, then $(3, 7)$ would be the unique optimal solution instead of $(5, 4)$, as illustrated in [Figure 10.4](#).

To find the minimum value of the x coefficient for which $(5, 4)$ remains optimal, we must have the slope of the objective function be no more than the slope of the constraint $3x + 2y = 23$, that is,

$$-\frac{3}{2} \geq -\frac{c_x}{5}.$$

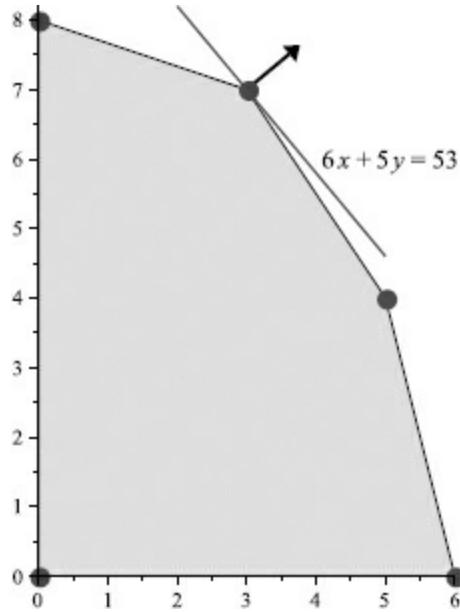
Doing this on the other side (to find the maximum value of the coefficient of x), we’d have the inequality

$$-4 \leq -\frac{c_x}{5},$$

giving the range of values for c_x if $(5, 4)$ is to remain optimal to be

$$7.5 \leq c_x \leq 20.$$

FIGURE 10.4 Optimal contour for objective function $z = 6x + 5y$.



Using the same approach for c_y , we'd find that the range of values for c_y if $(5, 4)$ is to remain optimal is

$$\frac{13}{4} \leq c_y \leq \frac{26}{3}.$$

In general, in linear programs with two variables, to find the range of values the objective coefficient can have, bound the slope of the objective function by the slopes of the binding constraints.

■ EXAMPLE 10.1

Suppose that the objective function in (10.1) had been $6x + 5y$, which results in $(3, 7)$ begin the optimal solution. For what objective coefficients c_y of y would $(3, 7)$ remain optimal? Since $(3, 7)$ occurs at the intersection of the constraints $3x + 2y = 23$ and $x + 3y = 24$, the slope of the contour line $6x + c_y y = k$ can range only between the slopes of the constraint lines, that is,

$$-\frac{3}{2} \leq -\frac{6}{c_y} \leq -\frac{1}{3}.$$

Thus, so long as $4 \leq c_y \leq 18$, the optimal solution to our linear program is $(3,$

7).

10.2 SENSITIVITY ANALYSIS CALCULATIONS

Can we generalize the work from the previous section to any linear program? After solving a mathematical program, we are often interested in knowing which changes could occur to the problem and still have the current “solution” remaining optimal. We have already seen that some changes to the linear program will alter the actual values of the variables. Since we’ve discussed the simplex method and we know what optimal solutions look like, we can clarify this question a bit more.

When doing sensitivity analysis for linear programs, we are trying to determine what effect changes in the linear program have on the optimal solution. In this regard, we are often interested in the possible changes to the linear program that allow the **basis**, or set of basic variables, to remain optimal.

By keeping the basis unchanged, all the information we gathered during the simplex method to formulate our basic feasible solution can be used with the new data, such as the matrices B and N , as well as the vectors \mathbf{c}_B and \mathbf{c}_N . To this end, we assume throughout this section that we have an optimal basic feasible solution $(\mathbf{x}_B, \mathbf{x}_N)$ to our maximization problem satisfying

$$(10.4) \quad \mathbf{x}_B = B^{-1}\mathbf{b} \geq \mathbf{0}, \quad \mathbf{x}_N = \mathbf{0}.$$

Reduced Costs How much must we adjust its objective coefficient of a nonbasic variable x_j in order for it to become basic in an optimal solution? Consider a maximization problem where an optimal basic feasible solution has the reduced costs

$$\bar{c}_k = \mathbf{c}^T \mathbf{d}^k = c_k - \mathbf{c}_B^T B^{-1} \mathbf{a}_k \leq 0$$

for every nonbasic variable x_k . If we were to increase c_k , the objective function coefficient of x_k , to $c_k + c_k + |\bar{c}_k|$, our new reduced costs computation would be $\bar{c}_k + |\bar{c}_k| = 0$. Our current basic feasible solution would still be

optimal, but there would be an alternative optimal solution with x_k being a basic variable (see Exercise 8.7). Now consider a minimization problem. At the optimal basic feasible solution, we know that $\bar{c}_k \geq 0$ for every nonbasic variable x_k . If we were to decrease c_k to $c_k - \bar{c}_k$, our new reduced costs computation would be $\bar{c}_k - \bar{c}_k = 0$. Again, our current basic feasible solution would still be optimal, but there would be an alternative optimal solution with x_k being a basic variable.

What about the basic variables? We earlier found that their reduced costs must be zero. Where does this occur in the calculations? For this, we need to look at an alternative view of the reduced cost calculations. From duality theory, we saw that the reduced cost computations

$$(10.5) \quad c_k - \mathbf{c}_B^T B^{-1} \mathbf{a}_k = c_k - \mathbf{y}^T \mathbf{a}_k$$

are nothing more than the difference between the left- and right-hand sides of each constraint in the dual, that is, how infeasible the current dual solution $\mathbf{y}^T = \mathbf{c}_B^T B^{-1}$ is. For each variable (basic and nonbasic), this value must be ≤ 0 in any feasible solution to the dual problem associated to a primal (maximization) problem in canonical form. By complementary slackness, if variable x_k is basic, (10.5) will have value 0.

■ EXAMPLE 10.2

Let's consider the linear program (10.1) that has the initial data (after putting the linear program in canonical form)

$$A = \begin{bmatrix} 4 & 1 & 1 & 0 & 0 \\ 1 & 3 & 0 & 1 & 0 \\ 3 & 2 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 13 \\ 5 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

At the optimal basic feasible solution $(5, 4, 0, 7, 0)$, the basis is $B = \{x, y, s_2\}$ with

$$B = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix} \text{ and } \mathbf{c}_B = \begin{bmatrix} 13 \\ 5 \\ 0 \end{bmatrix}.$$

This gives the optimal dual solution

$$\mathbf{y}^T = \mathbf{c}_B^T B^{-1} = \left[\frac{11}{5} \quad 0 \quad \frac{7}{5} \right].$$

Using (10.5), the reduced cost of basic variable x is

$$\begin{aligned}\bar{c}_x &= 13 - [13 \quad 5 \quad 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} \\ &= 13 - \left[\frac{11}{5} \quad 0 \quad \frac{7}{5} \right] \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} \\ &= 13 - \left(\frac{44}{5} + \frac{21}{5} \right) = 13 - \frac{65}{5} = 0.\end{aligned}$$

The reduced cost of the nonbasic variable s_3 would be

$$\begin{aligned}\bar{c}_{s_3} &= 0 - [13 \quad 5 \quad 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ &= 0 - \left[\frac{11}{5} \quad 0 \quad \frac{7}{5} \right] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\ &= -\frac{7}{5}.\end{aligned}$$

In order for s_3 to become basic at an optimal solution, its objective coefficient must increase by at least $|\bar{c}_{s_3}| = \frac{7}{5}$.

In terms of sensitivity analysis, the **Reduced Cost** of a variable gives information about how changes in the objective function coefficient of a given variable affects the optimal solution to the linear program. In general, the reduced cost is the amount that by which the objective function coefficient of a variable must be improved before the linear program will have an optimal solution with that variable having positive value. For a maximization problem, improving a coefficient means increasing it; for a minimization problem, improving a coefficient means decreasing it.

Shadow Prices In Sections 9.6 and 10.1, we defined the **shadow price** of a constraint as the amount by which the optimal value of the linear program would change per unit increase at the right-hand side. We have already seen that shadow prices are related to the values of the dual variables. Let's

explore this relationship more closely.

■ EXAMPLE 10.3

Consider again the linear program (10.1), but this time converted into canonical form, giving the initial data

$$A = \begin{bmatrix} 4 & 1 & 1 & 0 & 0 \\ 1 & 3 & 0 & 1 & 0 \\ 3 & 2 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 13 \\ 5 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 24 \\ 24 \\ 23 \end{bmatrix}.$$

The optimal basic feasible solution is $(5, 4, 0, 7, 0)$ and the optimal dual solution is $\mathbf{y} = \left(\frac{11}{5}, 0, \frac{7}{5}\right)$. The optimal value of both the primal problem and its dual is

$$\begin{aligned} z^* &= 13(5) + 5(4) \\ &= 24\left(\frac{11}{5}\right) + 23\left(\frac{7}{5}\right) \\ &= 85. \end{aligned}$$

Suppose that the right-hand side b_1 of the first primal constraint was changed from 24 to 25. Let's assume, furthermore, that the primal basis remains optimal, so that the optimal basic feasible solution to the primal corresponds to the basis $\{x, y, s_2\}$. In this case, B remains unchanged, which implies that the optimal dual solution is still $\mathbf{y}^* = \left(\frac{11}{5}, 0, \frac{7}{5}\right)$. The new optimal value to the dual problem would be

$$25\left(\frac{11}{5}\right) + 23\left(\frac{7}{5}\right) = 87\frac{1}{5} = 85 + \frac{11}{5}.$$

By the strong duality theorem, $87\frac{1}{5}$ would be the optimal value to the updated primal linear program. Hence, the shadow price associated to constraint 1 would be $\frac{11}{5}$. Using similar arguments, we can show that the shadow prices for constraints 2 and 3 would be 0 and $\frac{7}{5}$, respectively.

Note that, in a maximization problem where all (primal) variables are nonnegative, the value of a dual variable associated to a “ \leq ” constraint is always nonnegative, the value associated with a “ \geq ” constraint is always nonpositive, and the value associated to an equality constraint is unrestricted in sign. In terms of shadow prices, does this make sense? It should. If we

have a “ \leq ” constraint, and we increase the right-hand side value, we have a larger feasible region. Therefore, since the previous optimal solution remains feasible, **our new optimal value cannot decrease**. Hence, the shadow price must be nonnegative. For “ \geq ” constraint, increasing the right-hand side value results in a smaller feasible region, and because no new solutions are added to the feasible region, **our new optimal value cannot increase**. This implies that the shadow price must be nonpositive. For equality constraints, changing the right-hand side does not have a uniform effect on the feasible region—it neither necessarily enlarges nor contracts it; in addition, the current optimal solution is no longer feasible. Therefore, we cannot predict the sign of the shadow price.

What about for a minimization problem? Here, if all (primal) variables are again nonnegative, the value of a dual variable associated with a “ \leq ” constraint is nonpositive, the value associated with a “ \geq ” constraint is nonnegative, and the value associated with an equality constraint is unrestricted in sign. Now let’s look at the shadow prices for such a problem. If we have a “ \leq ” constraint and we increase the right-hand side, we are allowing more solutions to be feasible while retaining the previously feasible points. Hence, the optimal value must not increase, so a nonpositive shadow price is reasonable. Similarly for a “ \geq ” constraint, the shadow price would be nonnegative and an unknown sign for an equality constraint.

For a linear program that has a nondegenerate optimal solution, the shadow price of a constraint is equal to the optimal value of the dual variable associated with that constraint.

■ EXAMPLE 10.4

Consider the following linear program:

$$\begin{aligned} \min \quad & 13x - 5y \\ \text{s.t.} \quad & 4x + y \leq 24 \\ & x + 3y \leq 24 \\ & 3x + 2y \leq 23 \\ & x, y \geq 0. \end{aligned}$$

It is not difficult to see that the optimal solution to this linear program is $(0,$

8), while the optimal solution to its dual is $\mathbf{y}^* = (0, -\frac{5}{3}, 0)$. The optimal value of each linear program is -40 . If we were to increase the right-hand side of the second constraint from 24 to 25, the optimal solution to this new linear program would be $(0, \frac{25}{3})$ with optimal value $-5(\frac{25}{3}) = -\frac{125}{3} = -41\frac{2}{3}$. Note that the value changed by $-\frac{5}{3}$; hence, its shadow price is $-\frac{5}{3}$, the value of the dual variable corresponding to this constraint.

Objective Coefficient and Right-Hand-Side Ranges In Section 10.1, we said the **objective coefficient ranges** indicate the range of values that guarantee that the current variables with positive optimal value remain positive in an optimal solution. Similarly, in Section 10.1 we defined the **right-hand-side ranges** as the range of values (inclusive) of the right-hand-sides for which the shadow prices are valid. It also indicates the range of right-hand-side values for which the variables that are currently positive will remain positive in an optimal solution. We shall now find how these values are calculated.

We assume that our linear program is in canonical form

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ & \text{s.t.} \\ & \quad A \mathbf{x} = \mathbf{b} \\ & \quad \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

Let \mathbf{x}^* be an optimal basic feasible solution with basis matrix B whose columns correspond to the basic variables of \mathbf{x}^* and “let N be the matrix whose columns correspond” to the nonbasic variables. Since \mathbf{x}^* is optimal, its reduced costs are nonpositive, that is,

$$\bar{\mathbf{e}}^T = \mathbf{c}^T - \mathbf{c}_B^T B^{-1} A \leq \mathbf{0}^T,$$

where \mathbf{c}_B are the objective function coefficients associated to the basic variables. In fact, we know that, no matter what changes are made, we will always have

$$\bar{\mathbf{e}}_B^T = \mathbf{c}_B^T - \mathbf{c}_B^T B^{-1} B = \mathbf{0}^T.$$

Objective Function Coefficient Ranges Let’s first examine how to obtain ranges for the objective function coefficients.

■ EXAMPLE 10.5

Consider again the problem examined in Example 10.2. We had

$$B = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}, \quad N = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix},$$

$$\bar{e}_N^T = \begin{bmatrix} -\frac{11}{5} \\ -\frac{7}{5} \end{bmatrix}, \quad y^T = \begin{bmatrix} \frac{11}{5} & 0 & \frac{7}{5} \end{bmatrix}.$$

Suppose we want to find the largest change Δc_x to $c_x = 13$ so that the optimal solution remains $(5, 4, 0, 7, 0)$, implying that its reduced cost vector remains nonpositive. Since we only need to concern ourselves with the reduced costs of the nonbasic variables, we want to find a range of Δc_x where

$$\bar{e}_N^T = [0 \ 0] - [13 + \Delta c_x \ 5 \ 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\leq [0 \ 0].$$

This yields

$$\begin{aligned} \bar{e}_N^T &= \left([0 \ 0] - [13 \ 5 \ 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \right) \\ &\quad - \Delta c_x [1 \ 0 \ 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \left[-\frac{11}{5} \ -\frac{7}{5} \right] - \Delta c_x [1 \ 0 \ 0] \begin{bmatrix} \frac{2}{5} & -\frac{1}{5} \\ -\frac{3}{5} & \frac{4}{5} \\ \frac{7}{5} & -\frac{11}{5} \end{bmatrix} \\ &= \left[-\frac{11}{5} \ -\frac{7}{5} \right] - \Delta c_x [\frac{2}{5} \ -\frac{1}{5}] \\ &\leq [0 \ 0]. \end{aligned}$$

This implies that

$$-\frac{11}{5} - \frac{2}{5} \Delta c_x \leq 0 \quad \text{and} \quad -\frac{7}{5} + \frac{1}{5} \Delta c_x \leq 0,$$

yielding

$$-\frac{11}{2} \leq \Delta c_x \leq 7.$$

Thus, as long as

$$\frac{15}{2} \leq c_x \leq 20,$$

our current solution remains optimal.

Now suppose that we want to change the objective function coefficient of

the nonbasic variable s_1 from 0 to Δc_{s_1} . We would then want

$$\begin{aligned}\bar{\mathbf{c}}_N^T &= [\Delta c_{s_1} \ 0] - [13 \ 5 \ 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \\ &\leq [0 \ 0].\end{aligned}$$

Hence,

$$\begin{aligned}\bar{\mathbf{c}}_N^T &= \left([0 \ 0] - [13 \ 5 \ 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \right) + \Delta c_{s_1} [1 \ 0] \\ &= [-\frac{11}{5} \ -\frac{7}{5}] + \Delta c_{s_1} [1 \ 0] \\ &\leq [0 \ 0].\end{aligned}$$

This yields

$$\Delta c_{s_1} \leq \frac{11}{5}.$$

So long as $c_{s_1} \leq \frac{11}{5}$, our current solution remains optimal.

Let's generalize the computations given in the above example. Suppose that we want to change the value of the objective function coefficient of variable x_k from c_k to $c_k + \Delta c_k$. In terms of vectors, this implies that our new objective function coefficient vector would be $\hat{\mathbf{c}} = \mathbf{c} + \Delta c_k \mathbf{e}_k$, where \mathbf{e}_k is a unit vector with a 1 in the k th coordinate and 0 everywhere else. If we want to find values Δc_k for which the current basic variables remain basic in the new optimal solution, we need to ensure that

$$(10.6) \quad \hat{\mathbf{c}}_N^T - \hat{\mathbf{c}}_B^T B^{-1} N \leq \mathbf{0}^T,$$

Where $\hat{\mathbf{c}}_B$ and $\hat{\mathbf{c}}_N$ are the basic and nonbasic components of $\hat{\mathbf{c}}$ for the current set of basic variables. We break this up into two cases.

Case 1: x_k is a basic variable. In this case, (10.6) reduces to

$$\begin{aligned}\mathbf{c}_N^T - (\mathbf{c}_B + \Delta c_k \mathbf{e}_k)^T B^{-1} N &\leq \mathbf{0}^T \\ \iff (\mathbf{c}_N^T - \mathbf{c}_B^T B^{-1} N) - \Delta c_k \mathbf{e}_k^T B^{-1} N &\leq \mathbf{0}^T \\ (10.7) \iff \bar{\mathbf{c}}_N^T - \Delta c_k \mathbf{e}_k^T B^{-1} N &\leq \mathbf{0}^T.\end{aligned}$$

Let $\mathbf{d}^k = \mathbf{e}_k^T B^{-1} N$ (i.e., the k th row of $B^{-1} N$ if x_k is the k th basic variable in order). If the j th component of \mathbf{d}^k $d_j^k < 0$, then (10.7) becomes

$$\Delta c_k \leq \frac{-\bar{c}_j}{-d_j^k} = \frac{\bar{c}_j}{d_j^k}.$$

If $d_j^k > 0$, then (10.7) becomes

$$\Delta c_k \geq \frac{\bar{c}_j}{d_j^k}.$$

For each case regarding d_j^k , we must take care in case the vector is nonpositive or nonnegative. With this in mind, we have the following ranges of values:

$$\max \left\{ -\infty, \max_{d_j^k > 0} \left\{ \frac{\bar{c}_j}{d_j^k} \right\} \right\} \leq \Delta c_k \leq \min \left\{ \infty, \min_{d_j^k < 0} \left\{ \frac{\bar{c}_j}{d_j^k} \right\} \right\}.$$

Case 2: x_k is a nonbasic variable. In this case, (10.6) becomes

$$\begin{aligned} (\mathbf{e}_N^T + \Delta c_k \mathbf{e}_k^T) - \mathbf{e}_B^T B^{-1} N &\leq \mathbf{0}^T \\ \iff (\mathbf{e}_N^T - \mathbf{e}_B^T B^{-1} N) + \Delta c_k \mathbf{e}_k^T &\leq \mathbf{0}^T \\ \iff \bar{\mathbf{e}}_N^T + \Delta c_k \mathbf{e}_k^T &\leq \mathbf{0}^T. \end{aligned}$$

Since \mathbf{e}_k is nonzero in only the k th component, this reduces to

$$-\infty < \Delta c_k \leq -\bar{c}_k.$$

■ EXAMPLE 10.6

For the problem examined in Example 10.5, when changing c_x , if we use the formulas we just derived, we'd get

$$\mathbf{d} = \mathbf{e}_1^T B^{-1} N = [1 \quad 0 \quad 0] \begin{bmatrix} \frac{2}{5} & -\frac{1}{5} \\ -\frac{3}{5} & \frac{4}{5} \\ \frac{7}{5} & -\frac{11}{5} \end{bmatrix} = \begin{bmatrix} \frac{2}{5} & -\frac{1}{5} \end{bmatrix}.$$

Since $d_1 > 0$, we get

$$\Delta c_x \geq \frac{-(\frac{11}{5})}{(\frac{2}{5})} = -\frac{11}{2},$$

while $d_2 < 0$ implies

$$\Delta c_x \leq \frac{(-\frac{7}{5})}{(-\frac{1}{5})} = 7.$$

This yields

$$-\frac{11}{2} \leq \Delta c_x \leq 7.$$

Right-Hand-Side Ranges Now consider the effect of changing one

coefficient in the right-hand side. Suppose we are changing the right-hand-side value of the i th constraint from b_i to $b_i + b_i \Delta$, and hence the vector \mathbf{b} changes to $\mathbf{b} + b_i \mathbf{e}_i$, where again \mathbf{e}_i is a unit vector with a 1 in the i th component and 0 everywhere else. Currently, given the basis matrix B , our current optimal basic feasible solution is

$$\mathbf{x}^* = \begin{bmatrix} \mathbf{x}_B^* \\ \mathbf{x}_N^* \end{bmatrix} = \begin{bmatrix} B^{-1}\mathbf{b} \\ \mathbf{0} \end{bmatrix} \geq \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}.$$

If we change b_i , the basic solution with the same basic variables as above would be

$$\begin{bmatrix} B^{-1}(\mathbf{b} + \Delta b_i \mathbf{e}_i) \\ \mathbf{0} \end{bmatrix}.$$

When a change to b_i is made, if we want our current basic variables to remain basic in the new optimal solution, we must ensure that

$$(10.8) \quad \begin{bmatrix} B^{-1}(\mathbf{b} + \Delta b_i \mathbf{e}_i) \\ \mathbf{0} \end{bmatrix} \geq \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}.$$

It can be shown that the values of b_i that guarantees(10.8) holds satisfy

$$(10.9) \quad \max \left\{ -\infty, \max_{d_k > 0} \left\{ \frac{-(B^{-1}\mathbf{b})_k}{d_k} \right\} \right\} \leq \Delta b_i \leq \min \left\{ \infty, \min_{d_k < 0} \left\{ \frac{-(B^{-1}\mathbf{b})_k}{d_k} \right\} \right\},$$

where $\mathbf{d} = B^{-1} \mathbf{e}_i$ is the i th column of B^{-1} . Note that we would calculate \mathbf{d} as the solution to $B \mathbf{d} = \mathbf{e}_i$.

■ EXAMPLE 10.7

Continuing from Example 10.5, suppose we need to change the right-hand side of the first constraint from 24 to $24 + \Delta b_1$. To calculate the values of Δb_1 that keep the basis $\{x, y, s_2\}$ optimal, we first need to find \mathbf{d} by solving the system of equations

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \\ d_{s_2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Hence,

$$\mathbf{d} = \begin{bmatrix} \frac{2}{5} \\ -\frac{3}{5} \\ \frac{7}{5} \end{bmatrix}.$$

Since $\mathbf{x}_B = B^{-1} \mathbf{b} = (5, 4, 7)$, we have

$$\max \left\{ \frac{-5}{(\frac{2}{3})}, \frac{-7}{(\frac{7}{3})} \right\} = -5 \leq \Delta b_1 \leq \frac{20}{3} = \min \left\{ -\frac{4}{(-\frac{3}{3})} \right\}.$$

10.3 USE OF SENSITIVITY ANALYSIS

Now that we've seen how to calculate various sensitivity analysis values and ranges, it's time we see how these values can be used in post-solution modeling. We use the fact that commercial linear programming modeling packages can output all the sensitivity analysis information we need. In this section, we will discuss how to interpret such information by examining the output generated by the optimization software package Xpress-MP.¹

Most often, we will encounter nondegenerate solutions to our linear programs. However, degeneracy is something we must be concerned about when interpreting sensitivity analysis reports. We will look at the interpretation of the sensitivity analysis reports when there is a nondegenerate solution and a degenerate solution.

Sensitivity Analysis with Nondegenerate Solutions

Throughout this section, we examine the sensitivity reports of the two linear programs, one a maximization problem and the other a minimization problem, each of which has a nondegenerate solution. The optimal values of each linear program will be designated with a (*).

¹Many packages generate these types of results, so there is nothing unique about the output given in this section.

■ EXAMPLE 10.8

Wood Built Bookshelves (WBB) is a small wood shop that produces three types of bookshelves: models A, B, C. Each bookshelf requires a certain amount of time for cutting each component, assembling them, and then staining them. WBB also sells unstained versions of each model. The times,

in hours, for each phase of construction and the profit margins for each model, as well as the amount of time available in each department over the next two weeks, are given below.

Model	Cutting	Labor (h)		Profit Margin	
		Assembling	Staining	Stained	Unstained
A	1	4	7	\$60	\$30
B	0.5	3	5	\$40	\$20
C	2	6	8	\$75	\$40
Labor available	200	700	550		

Since this is the holiday season, WBF can sell every unit that it makes. Also, due to the previous year's demand, at least 20 model B bookshelves (stained or unstained) must be produced. Finally, to increase sales of the stained bookshelves, at most 50 unstained models are to be produced. How many of each model should be produced if WBF wants to maximize its holiday profits?

If we let A, B, C denote the number of stained models A, B, and C that are to be produced, and UA, UB, UC the number of unstained models, we would have the following linear program:

$$\begin{aligned}
 \max \quad & 60A + 40B + 75C + 30UA + 20UB + 40UC \\
 \text{s.t.} \quad & \\
 & (A + UA) + 0.5(B + UB) + 2(C + UC) \leq 200 \quad (\text{Cutting}) \\
 & 4(A + UA) + 3(B + UB) + 6(C + UC) \leq 700 \quad (\text{Assembling}) \\
 & 7A + 5B + 8C \leq 550 \quad (\text{Staining}) \\
 & UA + UB + UC \leq 50 \quad (\text{Unstained maximum}) \\
 & B + UB \geq 20 \quad (\text{B's production}) \\
 & A, B, C, UA, UB, UC \geq 0.
 \end{aligned}$$

The optimal solution found by Xpress-MP is $A^* = 30, B^* = 20, C^* = 30, UC^* = 50$ with value \$6850. The solution results generated from Xpress-MP are given below. Note that there are five constraints and five nonzero variables (including one slack variable).

Problem status: Optimum found

Optimal Profit = 6850

Variable	Activity	Cost	Reduced Coeff	(Lower, Upper)
A	30	0.0	60.00	(58.75, 65.63)
B	20	0.0	40.00	(32.50, 41.25)
C	30	0.0	75.00	(68.57, 77.14)
UA	0	-2.5	30.00	(-1e+020, 32.50)
UB	0	-7.5	20.00	(-1e+020, 27.50)
UC	50	0.0	40.00	(37.50, 1e+020)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
cutting	0	7.50	200.00	(174.29, 222.50)
assembling	40	0.00	700.00	(660.00, 1e+020)
staining	0	7.50	550.00	(460.00, 670.00)
Unstained	0	25.00	50.00	(38.75, 62.86)
B Production	0	-1.25	20.00	(-1e-006, 50.00)

■ EXAMPLE 10.9

Sycamore Basketball Company forecasts a 500-unit demand for its latest outdoor basketball hoop system during the next quarter. This hoop is assembled from three major components: support pole, backboard, and rim. Below are the production times and labor hours available. Note that each component produced must go through each department.

Department	Production times (h)			Time Available (h)
	Pole	Backboard	Rim	
A	2	2.5	1	2000
B	0.5	1	1.5	900
C	1	2	1	1500

Until now, Sycamore Basketball has manufactured all its components. However, it has never had such demand and is unsure if it can produce all 500. Thus, management has allowed for contracting the production of support poles or rims to a local firm. Below are the estimated costs for both manufacturing and purchasing each component.

Component	Manufacturing Cost	Purchase Cost
Pole	\$60	\$95
Backboard	\$80	—
Rim	\$30	\$45

Determine the make-or-buy policy for each component if Sycamore Basketball wants to minimize costs.

If we let $Pole M$, $Backboard M$, $Rim M$ denote how many of each component is manufactured and $Pole P$, $Rim P$ denote how many are purchased, our linear program would be

$$\begin{aligned}
\text{min} \quad & 60PoleM + 80BackboardM + 30RimM + 95PoleP + 45RimP, \\
\text{s.t.} \quad & \\
& 2PoleM + 2.5BackboardM + RimM \leq 2000 \quad (\text{Dept. A}) \\
& 0.5PoleM + BackboardM + 2RimM \leq 900 \quad (\text{Dept. B}) \\
& PoleM + 2BackboardM + RimM \leq 1500 \quad (\text{Dept. C}) \\
& PoleM + PoleP \geq 500 \quad (\text{Pole}) \\
& BackboardM \geq 500 \quad (\text{Backboard}) \\
& RimM + RimP \geq 500 \quad (\text{Rim}) \\
& \\
& PoleM, BackboardM, RimM, PoleP, RimP \geq 0.
\end{aligned}$$

The optimal solution is $Pole M^* = 375$, $Backboard M^* = 500$, $Rim M^* = 0$, $Pole P^* = 125$, and $Rim P^* = 500$, yielding an optimal cost of \$96,875. The XpressMP generated output is given below:

Problem status: Optimum found

Optimal value = 96875

Variable	Activity	Reduced Cost	Coeff	(Lower, Upper)
PoleM	375.00	0.0	60.00	(-1e+020, 65.00)
BackboardM	500.00	0.0	80.00	(-43.75, 1e+020)
RimM	0.00	2.5	30.00	(27.50, 1e+020)
PoleP	125.00	0.0	95.00	(90.00, 1e+020)
RimP	500.00	0.0	45.00	(-1e-006, 47.50)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
DeptA	0.0	17.50	2000.00	(1250.00, 2250.00)
DeptB	212.5	0.00	900.00	(687.50, 1e+020)
DeptC	125.0	0.00	1500.00	(1375.00, 1e+020)
Pole	0.0	95.00	500.00	(375.00, 1e+020)
Backboard	0.0	123.75	500.00	(400.00, 666.67)
Rim	0.0	45.00	500.00	(-1e-006, 1e+020)

Let us describe what each part of this output means in terms of our model. In each case, we assume that there are exactly m variables (including slack/surplus variables) that have positive value in the current optimal solution, where m is the number of constraints.

Reduced Costs Reduced costs give information about how changes in the objective function coefficient of a given variable affects the LP's optimal solution. In general, the reduced cost is the amount by which the objective function coefficient of a variable must be improved before the linear program will have an optimal solution with that variable having positive value. For a maximization problem, improving a coefficient means increasing it and for a minimization problem, improving a coefficient means decreasing it. If a variable has value zero, its reduced cost is the difference between the original

objective function coefficient and either the lower or upper range that corresponds to that variable (the other will be infinite). If we decrease the objective function coefficient by exactly the (nonzero) reduced cost, there will be alternative optima, one in which the given variable has nonzero value. Note that if a variable is already positive, its reduced cost is zero, unless it is at its upper bound (i.e., there exists a constraint of the form $x_i \leq u_i$ and $x^*_i = u_i$); in this case, the reduced cost is the amount needed to change the objective function coefficient in order to decrease the variable.

In Example 10.8, all decision variables have reduced costs of zero except for variables UA and UB , since these are the only variables with value zero. The reduced cost of UA is -2.5 , indicating that, in order for UA to have positive value in an optimal solution, its objective function coefficient must be improved (increased) by at least 2.5 to 32.50 ; similarly, the objective coefficient of UB must be at least 27.50 before UB can attain a positive value in an optimal solution. In Example 10.9, only $Rim\ M$ has value zero and a nonzero reduced cost. Its reduced cost of 2.5 implies that, in order for $Rim\ M$ to have positive value, its objective function coefficient must be improved (decreased) by at least 2.5 to 27.5 .

Objective Coefficient Ranges The objective coefficient ranges indicate the range of values (noninclusive) that guarantee that the current variables with positive optimal value remain positive. If the objective coefficient is changed to exactly either the lower range or upper range, then there exists at least one more optimal solution.

Consider Example 10.8. If we change the objective function coefficient of variable B to any value c_B satisfying $32.5 < c_B < 41.25$, then the current optimal solution remains optimal. Any change outside this range results in a new optimal solution. If $c_B = 32.5$ or $c_B = 41.25$, then the current solution will also remain optimal, but there will be a different solution with the same value; that is, there will be multiple optima. Note that, for variables UA and UB , the lower range for each is $-1E + 020$, which is treated as $-\infty$, while the upper ranges are exactly the current objective coefficient plus the reduced cost of that variable. Finally, take a look at variable UC . Its upper range is ∞ . Does this make sense? Yes, because UC is already at its upper limit, so increasing its profit margin by any amount does not alter its value.

Shadow Prices In cases where our optimal solution is not degenerate, the

shadow price of a constraint is the value of the constraint's dual variable. These amounts are valid only for right-hand-side values that fall within the RHS ranges. Note that if a constraint has slack or surplus, then by complementary slackness its shadow price must be zero.

In Example 10.8, the shadow price for the constraint on cutting is 7.5. Any unit increase in the right-hand side of 200 increases the optimal value of the linear program by 7.50. If we must produce more than 20 model B bookshelves, since the shadow price is -1.25 , we can expect our profits to decrease by \$1.25. If we were to increase the number of hours in the assembling department, our profit would not change (shadow price is zero). This should make sense since we are not using all the available hours in the original problem.

In Example 10.9, the shadow price for the constraint on Department A time is 17.5. If one more hour is made available to Department A, the total cost would decrease by \$17.50. If we were to require that at least 501 backboards be made available, it would raise our costs by \$123.75 since the shadow price for the backboard constraint is 123.75.

Right-Hand-Side Ranges The right-hand-side ranges indicate the values (inclusive) of the right-hand-sides for which the shadow prices are valid. It also indicates the range of right-hand-side values for which the variables that are currently positive will remain positive in an optimal solution.

For example, in Example 10.8, the RHS ranges for the staining constraint are $(460, 670)$. If the right-hand side of the staining constraint remains within this range, then (1) the variables that are currently positive (A, B, C, UC , and slack variable for assembling constraint) will remain positive in an optimal solution to the problem, and (2) the shadow price of 7.5 remains valid. If the right-hand side is not within this range, the shadow price is no longer valid (and will change), and at least one of the variables that are currently positive will have value zero in the new optimal solution. Note that, for the assembling constraint, the upper bound is ∞ . This should make sense since we do not use all the available labor in the cutting department, and so adding more should not change our answers at all (except for the slack variable corresponding to this constraint), no matter how much we add.

Examples of Sensitivity Analysis Interpretation

Let's now look at some types of questions that can be answered using sensitivity analysis.

1. In Example 10.8, by how much would we have to increase the profit margin of model B before we start making more than 20 units?

Answer: Since the objective range upper bound for B is 41.25, we'd have to increase the profit margin by at least $(\$41.25 - \$40) = \$1.25$ before we'd make anything other than 20 units. Since we're increasing the profit margin, it would make sense that more units would be made. However, we cannot be 100% sure of this until we resolve the problem.

2. In Example 10.9, how many poles are produced if each one costs \$62.50 to make? If they cost \$70?

Answer: Since the objective ranges for $PoleM$ are $(-\infty, 65)$, if each pole costs \$62.50, we would still make 375 because 62.50 is within the range. However, if the cost rises to \$70, this is outside the range, and hence we do not know, from the sensitivity reports, what the optimal solution would be. We would have to resolve the problem with the new cost.

3. In Example 10.8, if we sold unstained model B bookshelves for their original price of \$80, what is the production cost of such bookshelves that would allow us to optimally sell some?

Answer: Since the profit margin of unstained model B bookshelves is \$20, their production cost is \$60. We also know, by the reduced cost -7.5 of UB , that the profit margin would have to increase by \$7.5 before variable UB would have positive value in some optimal solution. Thus, if we do not change the selling price, we must decrease the cost in order to increase the profit margin. This implies that our production costs must be less than $(\$60 - \$7.5) = \$52.50$ per unit in order for us to produce these bookshelves.

4. Suppose that, in Example 10.9, Sycamore Basketball has an opportunity to hire 100 additional hours for Department A. What is the most they would be willing to pay for each hour?

Answer: Since adding 100 hours is within our RHS ranges, the shadow price of 17.5 is valid for this constraint. Each hour added decreases our cost by \$17.50, so the most they would be willing to spend is \$17.50 per hour.

Sensitivity Analysis and Degeneracy

When the optimal solution to a linear program is degenerate, the interpretation of any sensitivity reports must be done with care. Consider the following linear program

$$\begin{aligned}
 \max \quad & 3x + 8y + z \\
 \text{s.t.} \quad & \\
 & x + 4y + 5z \leq 20 \\
 & x + y + 3z \leq 9 \\
 & 2x + 3y + z \leq 20 \\
 & x + 2y + 2z \leq 12 \\
 (10.10) \quad & x, y, z \geq 0.
 \end{aligned}$$

The Xpress-MP generated output is given below:

Problem status: Optimum found

Optimal cost = 96875

Variable	Activity	Reduced Cost	Coeff	(Lower, Upper)
x	4	0.0	3.00	(2.00, 5.33)
y	4	0.0	8.00	(4.50, 12.00)
z	0	-6.8	1.00	(-1e+020, 7.80)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
eq1	0	1.4	20.00	(15, 20)
eq2	1	0	9.00	(8, 1e+020)
eq3	0	0.8	20.00	(15, 20)
eq4	0	0	12.00	(12, 1e+020)

Note that we have a degenerate optimal solution since only two of the original three variables and one of the slack variables are positive and there are four constraints. When degeneracy occurs, some unusual happenings occur in the sensitivity reports.

1. In at least one of the constraints, the RHS ranges has either a lower bound or an upper bound equal to the original right-hand side. This means that the shadow price is valid only for either an increase or a decrease of this right-hand side, but not both. In our example, this occurs with constraints 1, 3, and 4.
2. For some variable with value zero, its objective function coefficient may have to change by more than its reduced cost. In general, if a coefficient is changed in the appropriate direction by exactly its reduced cost, there exists a solution where that variable has positive value. When we have a

degenerate solution, this does not always happen. For example, we must change the objective coefficient of z to 9 before there is a solution where z is positive.

Thus, in the case where degeneracy exists at an optimal solution, we must be careful when trying to interpret the sensitivity analysis output.

10.4 PARAMETRIC PROGRAMMING

Now that we've considered what effect changing one parameter of our linear program has on our optimal solution, let's examine the effect of changing multiple parameters. Previously, we assumed that our current basic feasible solution remains optimal after the changes. Now, however, we will explore how various changes to the parameters alter our optimal solution.

In this section, we consider the linear program

$$\max \quad \mathbf{c}^T \mathbf{x} + \theta \mathbf{d}^T \mathbf{x}$$

s.t.

$$A\mathbf{x} = \mathbf{b}$$

$$(10.11) \quad \mathbf{x} \geq \mathbf{0},$$

where \mathbf{d} is some fixed vector and θ is a parameter that will change. For each value of θ , we get a linear program that we can solve; hence, we are interested in how our optimal solution changes as we change θ . Since we are altering only the parameter θ , this approach is called **parametric programming**.

To illustrate some of the fundamental ideas, let's consider the following example.

■ EXAMPLE 10.10

We will consider the parametric program

$$\begin{aligned}
\max \quad & 13x + 5y + \theta(-x + y) \\
\text{s.t.} \quad & 4x + y \leq 24 \\
& x + 3y \leq 24 \\
& 3x + 2y \leq 23 \\
(10.12) \quad & x, y \geq 0.
\end{aligned}$$

Its feasible region was previously given in [Figure 10.1](#), and its extreme points are

$$V = \{(0, 0), (0, 8), (3, 7), (5, 4), (6, 0)\}.$$

Since the feasible region is bounded, we know that, for every value of θ , an optimal solution exists at one or more extreme points. If we plug each of the extreme points into the objective function of (10.12), we get the following functions of θ :

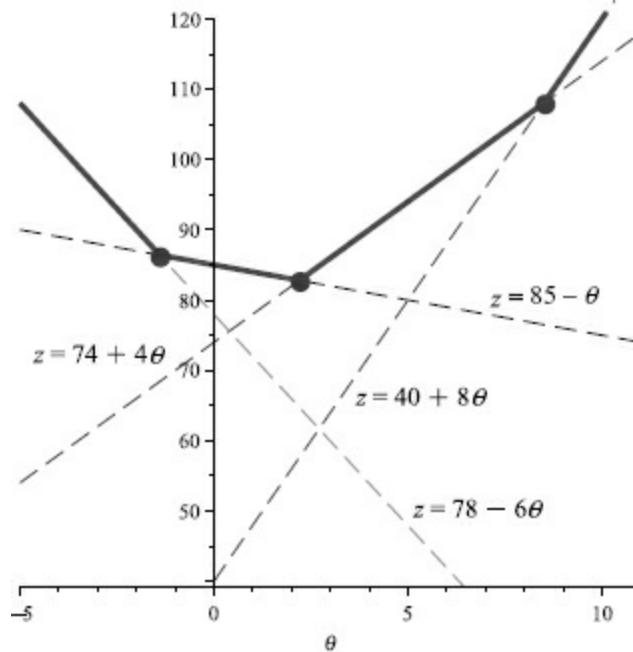
$$\begin{aligned}
z_{(0,0)} &= 0 \\
z_{(0,8)} &= 40 + 7\theta \\
z_{(3,7)} &= 74 + 4\theta \\
z_{(5,4)} &= 85 - \theta \\
z_{(6,0)} &= 78 - 6\theta.
\end{aligned}$$

For each value of θ , we are interested in finding

$$\begin{aligned}
z(\theta) &= \max \{z_{(0,0)}, z_{(0,8)}, z_{(3,7)}, z_{(5,4)}, z_{(6,0)}\} \\
&= \max \{0, 40 + 7\theta, 74 + 4\theta, 85 - \theta, 78 - 6\theta\}
\end{aligned}$$

In [Figure 10.5](#), we have each of the lines as dashed, and the function $z(\theta)$ as a solid curve. Note that $z(\theta)$ is piecewise linear and convex, where the breakpoints occur where there are multiple extreme points with the same optimal value; for example, when $\theta = \frac{11}{5}$, both the solutions $(5, 4)$ and $(3, 7)$ have the optimal Value $82\frac{4}{5}$.

FIGURE 10.5 Graph of $z(\theta)$ from Example 10.10.



Example 10.10 illustrates that if we have a bounded linear program and we know every extreme point, we can easily obtain the optimal value of (10.11) for any value of θ . What if we don't know the extreme points? We should be able to extend the work done in Section 10.2.

■ EXAMPLE 10.11

Consider again the linear program(10.11), but this time in the canonical form

$$\max \quad 13x + 5y + \theta(-x + y)$$

s.t.

$$4x + y + s_1 = 24$$

$$x + 3y + s_2 = 24$$

$$3x + 2y + s_3 = 23$$

$$x, y, s_1, s_2, s_3 \geq 0.$$

We have already seen in previous examples that, when $\theta = 0$, our optimal basic feasible solution is at $(5, 4, 0, 7, 0)$. If we now reintroduce θ into our objective function, we find that our reduced costs become

$$\begin{aligned} \bar{c}_N^T &= c_N^T - c_B^T B^{-1} N \\ &= [0 \ 0] - [13 - \theta \ 5 + \theta \ 0] \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \left[-\frac{11}{5} + \theta \ -\frac{7}{5} - \theta \right]. \end{aligned}$$

The basic feasible solution $(5, 4, 0, 7, 0)$ remains optimal as long as

$$\begin{aligned}-\frac{11}{5} + \theta &\leq 0 \\ -\frac{7}{5} - \theta &\leq 0,\end{aligned}$$

which gives us $-\frac{7}{5} \leq \theta \leq \frac{11}{5}$. What happens at the end points? Let's suppose that $\theta = \frac{11}{5}$. For this value of θ , the reduced cost $\bar{c}_{s_1} = 0$, indicating we have an alternative optimal solution. By letting s_1 be our entering variable in the simplex method, we would find that s_2 is our leaving variable, generating the new optimal basic feasible solution $(3, 7, 5, 0, 0)$ that remains optimal so long as its reduced cost vector

$$\begin{aligned}\bar{c}_N^T &= c_N^T - c_B^T B^{-1} N \\ &= [0 \ 0] - [13 - \theta \ 5 + \theta \ 0] \begin{bmatrix} 4 & 1 & 1 \\ 1 & 3 & 0 \\ 3 & 2 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= [-\frac{11}{7} - \frac{5}{7}\theta \ -\frac{34}{7} + \frac{4}{7}\theta]\end{aligned}$$

remains nonpositive; this occurs over the interval $\frac{11}{5} \leq \theta \leq \frac{17}{2}$. Now suppose that $\theta = -\frac{7}{5}$. For this value of θ , the reduced cost $\bar{c}_{s_3} = 0$, indicating that we have an alternative optimal solution. By letting s_3 be our entering variable in the simplex method, we would find that y is our leaving variable, generating the new optimal basic feasible solution $(6, 0, 0, 18, 5)$ that remains optimal so long as its reduced cost vector

$$\begin{aligned}\bar{c}_N^T &= c_N^T - c_B^T B^{-1} N \\ &= [5 + \theta \ 0] - [13 - \theta \ 0 \ 0] \begin{bmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ 3 & 0 \\ 2 & 0 \end{bmatrix} \\ &= [\frac{7}{4} + \frac{5}{4}\theta \ -\frac{13}{4} + \frac{1}{4}\theta]\end{aligned}$$

remains nonpositive; this occurs over the interval $-\infty < \theta \leq -\frac{7}{5}$. We can continue this analysis to solve(10.11) for all values of θ .

Notice that the procedure used in Example 10.11 can easily be generalized for any linear program. Care must be used if our parametric linear program (10.11) becomes unbounded for some value of θ (see Exercise 10.11), but for linear programs with bounded feasible regions, we can solve the parametric linear programming problem. The main observation is that, once we've fixed

our basic feasible solution, the reduced costs are all linear functions of θ plus some constant; thus, if we want all reduced costs to be nonpositive, θ must belong to some interval.

Solving the Parametric Programming Problem (10.11) with a Bounded Feasible Region

1. Solve linear program (10.11) for $\theta = 0$, generating an initial basic feasible solution \mathbf{x} .
2. Obtain the parametric programming reduced cost vector $\bar{\mathbf{c}}_N^\theta$ using the original objective function $(\mathbf{c} + \theta\mathbf{d})^T \mathbf{x}$. Solve the system of inequalities $\bar{\mathbf{c}}_N^\theta \leq \mathbf{0}$ to obtain an interval $\theta_L \leq \theta \leq \theta_U$ for which the current basic feasible solution is optimal.
3. At both $\theta = \theta_L$ and $\theta = \theta_U$, find the alternative optimal basic feasible solution using the simplex method. For each solution found, find the interval of θ for which it is optimal.
4. Repeat steps 2 and 3 until all values of θ have been included in some interval.

A similar problem can be formulated where the right-hand-side vector \mathbf{b} is changed to $\mathbf{b} + \theta\mathbf{d}$ and the objective coefficient vector \mathbf{c} is unchanged. In this problem, we need to worry about maintaining feasibility of our current basic solution. We leave this as exercises (see Exercises 10.12–10.14).

Summary

In this chapter, we explored how changes to parameters of our linear program affect the optimal basic feasible solution and its optimal value. These changes can readily be calculated using the basis information generated by the simplex method. In addition, these values have economic interpretations relating to the original problem. This sensitivity analysis allows us to consider minor changes to our problem and examine their effects. Because parameters are often just approximations of their real values, this analysis often proves useful when we make decisions based upon the linear program results.

Sensitivity analysis and parametric programming provide one example of

the information that we can obtain from the optimal primal and dual solutions. In Chapter 11, we explore other uses of the dual problem, providing further evidence of its importance to optimization problems.

EXERCISES

10.1 Consider the linear program

$$\begin{aligned} \max \quad & 9x + 10y \\ \text{s.t.} \quad & x + 2y \leq 30 \\ & 3x + 2y \leq 42 \\ & 3x - y \leq 24 \\ & x, y \geq 0, \end{aligned}$$

whose optimal solution is $(x, y) = (6, 12)$.

- (a)** Use the graphical techniques from Section 10.1 to obtain the range of objective coefficient values for $(6, 12)$ to remain optimal.
- (b)** Use the graphical techniques from Section 10.1 to obtain the shadow prices of each constraint and the range of right-hand-side values for these shadow prices to remain valid.
- (c)** Verify the calculations of parts (a) and (b) by using the matrix calculations in Section 10.2.
- (d)** How is the optimal solution affected if we were to remove the constraint $3x - y \leq 24$ from the problem? What if we removed the constraint $x + 2y \leq 30$?

10.2 Consider the linear program

$$\begin{aligned} \max \quad & 5x - y \\ \text{s.t.} \quad & -x + 2y \leq 30 \\ & x + y \leq 27 \\ & 4x - y \leq 43 \\ & x - y \leq 7 \\ & x, y \geq 0, \end{aligned}$$

whose optimal solution is $(x, y) = (14, 13)$.

- (a)** Use the graphical techniques from Section 10.1 to obtain the range of objective coefficient values for $(14, 13)$ to remain optimal.

- (b)** Use the graphical techniques from Section 10.1 to obtain the shadow prices of each constraint and the range of right-hand-side values for these shadow prices to remain valid.
- (c)** Verify the calculations of parts (a) and (b) by using the matrix calculations in Section 10.2.
- (d)** How is the optimal solution affected if we were to remove the constraint $4x - y \leq 43$ from the problem? What if we removed the constraint $-x + 2y \leq 30$?
- (e)** How is the optimal solution affected if we add the constraint $5x + y \leq 80$?

10.3 Q Electronics (QE) manufactures several products, including the Q75-1 DVD player that can hold a single disc and the Q100-5 DVD player that can hold up to five discs. Because their products are of such high quality, QE can sell any DVD players they intend to produce. For each Q75-1 produced, QE makes \$55 in profit, while Q100-5 has a unit profit margin of \$75. Each DVD player produced must go through a production area, an assembly area, and a testing area. During each 8-hour shift, QE has available 385, 240, and 700 hours for production, assembly, and testing, respectively. Below are the time requirements, in hours, for each product in these areas.

	Production	Assembly	Testing
Q75-1	2	1	5
Q100-5	3	2	4

To maximize its profit, QE has formulated the following linear program:

$$\begin{aligned}
 & \text{max} && 55Q75 + 75Q100 \\
 & \text{s.t.} && \\
 & & 2Q75 + 3Q100 \leq 385 & \text{(Production)} \\
 & & Q75 + 2Q100 \leq 240 & \text{(Assembly)} \\
 & & 5Q75 + 4Q100 \leq 700 & \text{(Testing)} \\
 & & Q75, Q100 \geq 0,
 \end{aligned}$$

which when solved generated the following output:

Problem status: Optimum found

Optimal Profit = 10025

Variable	Activity	Reduced Cost	Obj. Coeff	(Lower, Upper)
	Q75	80	0	55.00 (50, 93.75)
	Q100	75	0	75.00 (44, 82.50)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
Production	0	22.143	385.00	(280, 396.67)
Assembly	10	0.000	240.00	(230, 1e+020)
Testing	0	2.143	700.00	(630, 962.50)

Answer each of the following questions using the information given in the above output. If you can only answer a question by resolving the linear program, you must indicate this.

- (a)** How many of each DVD player should QE produce to maximize its profit?
- (b)** Can QE charge more for its players without changing its production schedule? If so, by how much?
- (c)** QE is willing to increase the number of hours for production? How many hours of production should they add and how much should they be willing to pay for these extra hours?
- (d)** If QE were to increase the number of hours in testing by 10 hours, how will their profits change? By 20 hours?
- (e)** Management is not happy about the production amounts; they believe they are too unbalanced. They would like the number of Q100-5 players to be at least 10 more than the number of Q75-1 players produced. How does this affect the production schedule? Which areas (production, assembly, or testing) are now restricting the amounts produced?

10.4 BetterBall Inc. produces basketballs for various sporting good stores throughout central Indiana. It produces two sizes: a regulation men's ball and a regulation women's ball that is slightly smaller. Each ball is made of rubber and leather, with the men's ball requiring (taking into account waste) 300 square inches of leather and 275 square inches of rubber and the women's ball requiring 275 square inches of leather and 250 square inches of rubber. For the coming week, BetterBall has available 160,000 square inches of rubber and 150,000 square inches of leather. Each ball produced goes through two production areas: assembly and testing. The men's ball takes 30 minutes in assembly and 30 minutes in testing, while

the women's ball requires 24 and 30 minutes in these respective areas. There are currently 260 hours available for assembly. To satisfy union requirements, at least 275 hours must be spent in testing these balls. The profit margins on these balls are \$25 for men's balls and \$20 for women's balls. To maximize their profit, BetterBall solves the following linear program

$$\begin{aligned}
 \text{max } & 25\text{mens} + 20\text{womens} \\
 \text{s.t. } & \\
 & 300\text{mens} + 275\text{womens} \leq 160000 \quad (\text{leather}) \\
 & 275\text{mens} + 250\text{womens} \leq 150000 \quad (\text{rubber}) \\
 & 30\text{mens} + 24\text{womens} \leq 15600 \quad (\text{assembly}) \\
 & 30\text{mens} + 30\text{womens} \geq 16500 \quad (\text{testing}) \\
 & \text{mens, womens} \geq 0,
 \end{aligned}$$

where the right-hand-side values in the assembly and testing constraints are in minutes. The following output was generated:

Problem status: Optimum found

Optimal Profit = 12750

Variable	Activity	Reduced Cost	Obj. Coeff	(Lower, Upper)
<hr/>				
mens	350	0	25.00	(21.82, 1e+020)
womens	200	0	20.00	(-1e+020, 22.92)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
<hr/>				
leather	0	0.2	160000.0	(151250, 161250)
rubber	3750	0	150000.0	(146250, 1e+020)
assembly	300	0	15600.0	(15300, 1e+020)
testing	0	-1.167	16500.0	(16285.7, 17454.5)

Answer each of the following questions using the information given in the above output. If you can only answer a question by resolving the linear program, you must indicate this.

- (a)** What is the optimal production of basketballs for the current week?
- (b)** BetterBall has just received word from their supplier that 1000 square feet of leather can be purchased today at the reduced price of \$150 and 2500 square feet of rubber is available for \$200. Should BetterBall purchase any of these materials?
- (c)** Management notices that not every hour available for assembly is being used and is thinking of investing more time in testing. Would you

recommend this action? Explain.

(d) Marketing has just finished a study that indicates customers will continue to buy the women's basketball even if the price is increased by \$10 (which would increase the profit by that amount as well). From the information provided above, would the overall profit increase by \$2000 if this action were taken? Explain.

10.5 The Security Door Company designs three types of steel doors: Standard, High Security, and Maximum Security. Each door requires different amounts of machine (and labor) time and they also have different revenue; this information is given in the following table:

	Machine 1 Hours	Machine 2 Hours	Revenue
Standard	3	4	\$540
High Security	7	5	\$760
Maximum Security	9	7	\$950

Each door must go through both machine 1 and machine 2 before it can be sold. SDC has 350 hours per week available on machine 1 at a cost of \$30 per hour and 300 hours per week on machine 2 at a cost of \$40 per hour. In addition, SDC can get overtime hours on each machine at a cost of \$65 per hour on machine 1 and \$90 per hour on machine 2. The company has a known demand of at least 25 Standard doors, 25 High Security doors, and 10 Maximum Security doors for the upcoming week, although it anticipates additional demands and expects to sell anything it produces. It is hoping to generate over \$45,000 in revenue from the sales of the doors while minimizing total costs; to do so, it solves the following linear program:

$$\begin{aligned}
 \text{min} \quad & 30M1 + 40M2 + 65OT1 + 90OT2 \\
 \text{s.t.} \quad & 3Standard + 7HighSec + 9MaxSec = M1 + OT1 \quad (\text{Machine 1}) \\
 & 4Standard + 5HighSec + 7MaxSec = M2 + OT2 \quad (\text{Machine 2}) \\
 & M1 \leq 350 \quad (\text{Machine 1 Time}) \\
 & M2 \leq 300 \quad (\text{Machine 2 Time}) \\
 & 540Standard + 760HighSec + 950MaxSec \geq 45000 \quad (\text{Revenue}) \\
 & Standard \geq 25 \quad (\text{Standard Doors}) \\
 & HighSec \geq 25 \quad (\text{High Security Doors}) \\
 & MaxSec \geq 10 \quad (\text{Max Security Doors}) \\
 & Standard, HighSec, MaxSec, M1, M2, OT1, OT2 \geq 0.
 \end{aligned}$$

The following output was generated:

Problem status: Optimum found

Optimal Cost = 24483.3

Variable	Activity	Reduced Cost	Obj Coeff	(Lower, Upper)
<hr/>				
Standard	30.56	0.00	0.00	(-555.00, 88.03)
HighSec	25.00	0.00	0.00	(-123.89, 1e+020)
MaxSec	10.00	0.00	0.00	(-238.61, 1e+020)
M1	350.00	0.00	30.00	(-1e+020, 65.00)
M2	300.00	0.00	40.00	(-1e+020, 90.00)
OT1	6.67	0.00	65.00	(30.00, 1e+020)
OT2	17.22	0.00	90.00	(40.00, 286.76)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
<hr/>				
machine1	0.00	-65.00	0.00	(-1e+020, 6.67)
machine2	0.00	-90.00	0.00	(-1e+020, 17.22)
M1Time	0.00	-35.00	350.00	(0.00, 356.67)
M2Time	0.00	-50.00	300.00	(0.00, 317.22)
Revenue	0.00	1.03	45000.00	(43800.00, 1e+020)
NumStandard	-5.56	0.00	25.00	(-1e+020, 30.56)
NumHigh	0.00	123.89	25.00	(22.60, 28.95)
NumMax	0.00	238.61	10.00	(8.21, 13.16)

Answer each of the following questions using the information given in the above output. If you can only answer a question by resolving the linear program, you must indicate this.

- (a)** How much revenue was generated by the optimal solution?
- (b)** Suppose that the per-hour cost on machine 1 is needed to increase by \$10 due to maintenance problems. What is the optimal door production now? How much does it now cost?
- (c)** Suppose we receive an order for three Maximum Security doors that needs to be filled this week. How much can we expect our costs to increase?
- (d)** Management wants our weekly revenue to be \$50,000 instead of \$45,000. Would this result in producing more than 10 Maximum Security doors if we still want to minimize costs? Explain.
- (e)** Management wants our weekly revenue to be \$50,000 instead of \$45,000. Would this result in more profit for SDC? Explain.
- (f)** Explain in words why the dual value of machine 1 is -65.
- (g)** Explain in words why the dual value of constraint “M1Time” is -35.

10.6 In Example 2.5, we derived the following linear program for an inventory model.

$$\begin{aligned}
 \min \quad & 150 (IT_1 + IT_2 + IT_3 + IC_1 + IC_2 + IC_3) \\
 & + 2000 (T_1 + T_2 + T_3) + 1500 (C_1 + C_2 + C_3) \\
 \text{s.t.} \quad & \\
 & T_1 + C_1 \leq 1000 \quad (\text{Production}) \\
 & T_2 + C_2 \leq 1000 \\
 & T_3 + C_3 \leq 1000 \\
 & 10T_1 + 8C_1 \leq 9000 \quad (\text{Labor}) \\
 & 10T_2 + 8C_2 \leq 9000 \\
 & 10T_3 + 8C_3 \leq 9000 \\
 & IT_1 = 100 + T_1 - 400 \quad (\text{Inventory}) \\
 & IT_2 = IT_1 + T_2 - 300 \\
 & IT_3 = IT_2 + T_3 - 500 \\
 & IC_1 = 200 + C_1 - 800 \\
 & IC_2 = IC_1 + C_2 - 500 \\
 & IC_3 = IC_2 + C_3 - 600 \\
 & IC_3 \geq 100, IT_3 \geq 100 \quad (\text{End Inventory}) \\
 & T_1, T_2, T_3, C_1, C_2, C_3, \\
 & IT_1, IT_2, IT_3, IC_1, IC_2, IC_3 \geq 0.
 \end{aligned}$$

The output generated by solving this problem is given below:

Problem status: Optimum found

Optimal Cost = 5190000.00

Variable	Activity	Cost	Reduced Coeff	Obj (Lower, Upper)
C(1)	600.00	0.00	1500.00	(1500.00, 1e+020)
T(1)	400.00	0.00	2000.00	(1850.00, 2000.00)
C(2)	500.00	0.00	1500.00	(1500.00, 1500.00)
T(2)	500.00	0.00	2000.00	(2000.00, 2000.00)
C(3)	700.00	0.00	1500.00	(-450.00, 1500.00)
T(3)	300.00	0.00	2000.00	(2000.00, 2300.00)
IC(1)	0.00	0.00	150.00	(-150.00, 1e+020)
IT(1)	100.00	0.00	150.00	(-0.00, 150.00)
IC(2)	0.00	0.00	150.00	(-150.00, 1e+020)
IT(2)	300.00	0.00	150.00	(-150.00, 150.00)
IC(3)	100.00	0.00	150.00	(-1800.00, 1e+020)
IT(3)	100.00	0.00	150.00	(-2300.00, 1e+020)

Constraint	Slack	Dual Value	RHS	(Lower, Upper)
Production(1)	0.00	0.00	1000.00	(1000.00, 1e+020)
Production(2)	0.00	-150.00	1000.00	(1000.00, 1000.00)
Production(3)	0.00	-300.00	1000.00	(1000.00, 1040.00)
Labor(1)	200.00	0.00	9000.00	(8800.00, 1e+020)
Labor(2)	0.00	0.00	9000.00	(9000.00, 1e+020)
Labor(3)	400.00	0.00	9000.00	(8600.00, 1e+020)
Inv(Car,1)	0.00	-1500.00	-600.00	(-600.00, 0.00)
Inv(Truck,1)	0.00	-2000.00	-300.00	(-300.00, 100.00)
Inv(Car,2)	0.00	-1650.00	-500.00	(-500.00, -500.00)
Inv(Truck,2)	0.00	-2150.00	-300.00	(-300.00, -200.00)
Inv(Car,3)	0.00	-1800.00	-600.00	(-600.00, -500.00)
Inv(Truck,3)	0.00	-2300.00	-500.00	(-500.00, -400.00)
MinEnd(Car)	0.00	1950.00	100.00	(-0.00, 100.00)
MinEnd(Truck)	0.00	2450.00	100.00	(-0.00, 100.00)

Answer each of the following questions using the information given in the above output. If you can only answer a question by resolving the linear program, you must indicate this.

(a) Suppose we are able to increase the number of labor hours available in any one month. For which month (if any) should we increase labor hours and by how much? Explain.

(b) Suppose we are able to increase the number of production limits in any 1 month. For which month (if any) should we increase production and by how much? Explain.

(c) Suppose our demand for cars decreases during month 3 to 550 cars. How does this affect our total cost? What if it decreases to 450 cars?

10.7 Given the feasible region $\{(x, y) : x - y = 0, x \geq 0, y \geq 0\}$, show that the bounds $x \geq 0$ and $y \geq 0$ are both redundant, but that both cannot be removed without altering the feasible region.

10.8 Consider linear program (10.10) in which we saw that the optimal solution was degenerate. The objective coefficient range for variable z was

given as $(-\infty, 7.80)$. Show that z will not have positive value in any optimal solution until its coefficient is at least 9 by first setting its objective coefficient to 8 and observing its coefficient range.

10.9 Solve the parametric programming problem

$$\max \quad 9x + 10y + \theta(2x + y)$$

s.t.

$$x + 2y \leq 30$$

$$3x + 2y \leq 42$$

$$3x - y \leq 24$$

$$x, y \geq 0.$$

For each value of θ , indicate the optimal solution and its corresponding value.

10.10 Solve the parametric programming problem

$$\max \quad 3x + 2y + \theta(x + 3y)$$

s.t.

$$2x - y \leq 6$$

$$2x + y \leq 10$$

$$x, y \geq 0.$$

For each value of θ , indicate the optimal solution and its corresponding value.

10.11 Solve the parametric programming problem

$$\max \quad -x - y + \theta(2x + 3y)$$

s.t.

$$7x + 2y \geq 28$$

$$2x + 12y \geq 24$$

$$x, y \geq 0.$$

How do you know when the parametric problem is unbounded?

10.12 Consider the parametric programming problem

$$\max \quad 13x + 5y$$

s.t.

$$4x + y + s_1 = 24 + \theta$$

$$x + 3y + s_2 = 24 + 2\theta$$

$$3x + 2y + s_3 = 23 + 3\theta$$

$$x, y, s_1, s_2, s_3 \geq 0.$$

Solve this problem for all values of θ for which there exists a feasible solution.

10.13 Consider the parametric programming problem

$$\begin{aligned} \max \quad & x_1 + 3x_2 \\ \text{s.t.} \quad & \\ & x_1 + x_2 \geq 3 + \theta \\ & x_1 - x_2 \geq 1 - \theta \\ & x_1 + 2x_2 \leq 4 + 2\theta \\ & x_1, x_2 \geq 0 \end{aligned}$$

Solve this problem for all values of θ for which there exists a feasible solution.

10.14 Consider the parametric programming problem

$$\begin{aligned} \max \quad & x + y \\ \text{s.t.} \quad & \\ & -2x + y \leq 0 + 2\theta \\ & x - 2y \leq 0 + 3\theta \\ & x + y \leq 9 + \theta \\ & x, y \geq 0. \end{aligned}$$

Solve this problem for all values of θ for which there exists a feasible solution.

CHAPTER 11

ALGORITHMIC APPLICATIONS OF DUALITY

In the past few chapters, we've seen how general linear programs can be solved using the simplex method, and we took a different view of linear programs via duality. With this, we've seen how the simplex method can be viewed in completely different ways. Why is this important? Many algorithms, especially specialized versions of the simplex method for various classes of problems, use duality to get better implementations of the simplex method. In this chapter, we examine how duality is used to improve how the simplex method solves certain problems and to derive additional algorithms for linear programs.

11.1 DUAL SIMPLEX METHOD

In Chapter 9, we mentioned that the simplex method can be viewed as an algorithm that maintains primal feasibility and complementary slackness while trying to obtain dual feasibility. This is done by solving the primal problem, and when it finishes a solution to the dual problem is available. In this section, we consider the case where we have a feasible solution to the dual problem that satisfies complementary slackness along with an infeasible primal solution.

Throughout this section, we consider the linear program in canonical form

$$\begin{aligned} \max \quad & 13x + 5y \\ \text{s.t.} \quad & 4x + y + s_1 = 24 \\ & x + 3y + s_2 = 24 \\ & 3x + 2y + s_3 = 23 \\ (11.1) \quad & x, y, s_1, s_2, s_3 \geq 0 \end{aligned}$$

We have seen in Chapters 8 and 10 that the optimal basic feasible solution is $(5, 4, 0, 7, 0)$ with basis $B = \{x, y, s_2\}$ and basis matrix

$$B = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 3 & 2 & 0 \end{bmatrix}.$$

Its dual problem is

$$\begin{aligned} \min \quad & 24y_1 + 24y_2 + 23y_3 \\ \text{s.t.} \quad & 4y_1 + y_2 + 3y_3 \geq 13 \\ & y_1 + 3y_2 + 2y_3 \geq 5 \\ & y_1 \geq 0 \\ & y_2 \geq 0 \\ & y_3 \geq 0. \end{aligned}$$

Note that the constraints $y_k \geq 0$ in the dual are generated through $A^T \mathbf{y} \geq \mathbf{b}$ since the equality constraints in the primal (11.1) would normally indicate that each y_k is unrestricted in sign. The simplex method obtained the dual optimal solution $\mathbf{y} = (\frac{11}{3}, 0, \frac{7}{3})$.

Suppose, we add the constraint $14x + 6y \leq 89$ to (11.1), which, after inserting a slack variable s_4 , yields the linear program

$$\begin{aligned} \max \quad & 13x + 5y \\ \text{s.t.} \quad & 4x + y + s_1 = 24 \\ & x + 3y + s_2 = 24 \\ & 3x + 2y + s_3 = 23 \\ & 14x + 6y + s_4 = 89 \\ (11.2) \quad & x, y, s_1, s_2, s_3, s_4 \geq 0 \end{aligned}$$

Note that our current solution $(5, 4, 0, 7, 0, -5)$ is no longer feasible ($s_4 < 0$), but it is a basic solution with basis $\{x, y, s_2, s_4\}$ since the basis matrix

$$B = \begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 3 & 2 & 0 & 0 \\ 14 & 6 & 0 & 1 \end{bmatrix}$$

has full row rank.

How can we find an optimal solution to this problem? One approach would be to simply resolve the problem from scratch using the simplex method. This seems to be overkill since the new solution is probably “close” to our

current basic solution $(5, 4, 0, 7, 0, -5)$. We could also attempt to modify an improving search method to incorporate infeasible solutions.

However, given our knowledge of linear programming duality, perhaps we should examine the effect adding this constraint has on the dual problem. The dual problem, after adding the constraint to the primal, becomes

$$\begin{aligned}
 \min \quad & 24y_1 + 24y_2 + 23y_3 + 89y_4 \\
 \text{s.t.} \quad & \\
 & 4y_1 + y_2 + 3y_3 + 14y_4 \geq 13 \\
 & y_1 + 3y_2 + 2y_3 + 6y_4 \geq 5 \\
 & y_1 \geq 0 \\
 & y_2 \geq 0 \\
 & y_3 \geq 0 \\
 (11.3) \quad & y_4 \geq 0.
 \end{aligned}$$

Note that the addition of a constraint in the primal adds a variable y_4 in the dual. Also note that if we set $y_4 = 0$ and maintain the original values of y_1, y_2, y_3 , we get a feasible solution $(\frac{11}{5}, 0, \frac{7}{5}, 0)$. In fact, if we use the basis information from the primal (infeasible) basic solution, we have

$$\begin{aligned}
 \mathbf{y}^T &= \mathbf{c}_B^T \mathbf{B}^{-1} \\
 &= [13 \ 5 \ 0 \ 0] \begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 3 & 2 & 0 & 0 \\ 14 & 6 & 0 & 1 \end{bmatrix}^{-1} \\
 &= \left[\frac{11}{5} \ 0 \ \frac{7}{5} \ 0 \right].
 \end{aligned}$$

Since we have a feasible dual solution, what if we solve the dual problem using the simplex method and then use this to generate our primal solution? Before we begin, though, we need to first rewrite (11.3) into one with equality constraints by adding surplus variables w_j to each of the constraints:

$$\begin{aligned}
\min \quad & 24y_1 + 24y_2 + 23y_3 + 89y_4 \\
\text{s.t.} \quad & \\
& 4y_1 + y_2 + 3y_3 + 14y_4 - w_x = 13 \\
& y_1 + 3y_2 + 2y_3 + 6y_4 - w_y = 5 \\
& y_1 - w_{s1} = 0 \\
& y_2 - w_{s2} = 0 \\
& y_3 - w_{s3} = 0 \\
& y_4 - w_{s4} = 0 \\
(11.4) \quad & w_x, w_y, w_{s1}, w_{s2}, w_{s3}, w_{s4} \geq 0.
\end{aligned}$$

Note that we indexed the surplus variable by the corresponding primal variable that generated that constraint. In addition, we left the constraints $y_k \geq 0$ as constraints instead of treating them as nonnegativity bounds because the y_k variables were initially unrestricted. If we can find a basic feasible solution to (11.4) that has each of the y_k variables basic, then, because they are unrestricted in sign, they would never become nonbasic (since they would never be part of any ratio test, which we used to maintain nonnegativity of all (nonnegative) variables). We shall denote all matrices and vectors used in the simplex method by \hat{A} , \hat{B} , and so on, to indicate they come from the dual problem; thus, the matrix B corresponds to the primal basis matrix, while the matrix \hat{B} corresponds to the dual basis matrix.

The constraint matrix \hat{A} of (11.4) is

$$\hat{A} = \begin{bmatrix} 4 & 1 & 3 & 14 & -1 & 0 & 0 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Consider the basis matrix \hat{B} with basis $\{y_1, y_2, y_3, y_4, w_{s1}, w_{s3}\}$

$$\hat{B} = \begin{bmatrix} 4 & 1 & 3 & 14 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

This matrix has full row rank and, with right-hand-side vector \mathbf{c} of the dual, we get the system

$$\hat{B} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ w_{s1} \\ w_{s3} \end{bmatrix} = \mathbf{c},$$

which generates the solution

$$\begin{aligned} & \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ w_{s1} \\ w_{s3} \end{bmatrix} = \hat{B}^{-1} \mathbf{c} \\ &= \begin{bmatrix} 4 & 1 & 3 & 14 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 13 \\ 5 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \frac{11}{5} \\ 0 \\ \frac{7}{5} \\ 0 \\ \frac{11}{5} \\ \frac{7}{5} \end{bmatrix}. \end{aligned}$$

Given the primal basic solution, we can generate a basic feasible solution to the dual problem using the (unrestricted) variables y_k as part of the basis.

To solve the dual problem (11.4) using the simplex method, we first need to compute the reduced cost vector $\hat{\mathbf{c}}$ for the nonbasic variables w_x, w_y, w_{s2}, w_{s4} . Given our corresponding nonbasic matrix

$$\hat{N} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

and the fact that the objective coefficient for each of these variables is 0, we have

$$\hat{\bar{c}}^T = [0 \ 0 \ 0 \ 0]$$

$$- [24 \ 24 \ 23 \ 89 \ 0 \ 0] \begin{bmatrix} 4 & 1 & 3 & 14 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$= - [5 \ 4 \ 0 \ 7 \ 0 \ -5] \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$= [5 \ 4 \ 7 \ -5],$$

which are the values of the basic variables in the primal basic solution. Note that our multipliers in the next-to-last step are exactly the values of all primal variables.

Since we have a negative reduced cost in our minimization problem (corresponding to variable w_{s4}), our current dual basic feasible solution is not optimal (of course, by duality theory, we already knew this, since the corresponding primal problem is not feasible). We next find the simplex direction \mathbf{d} by solving

$$\hat{B}\mathbf{d} = -\hat{\mathbf{a}}_{w_{s4}},$$

which gives the direction

$$\mathbf{d} = \begin{bmatrix} d_{y_1} \\ d_{y_2} \\ d_{y_3} \\ d_{y_4} \\ d_{w_{s1}} \\ d_{w_{s3}} \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ -2 \\ 1 \\ -2 \\ -2 \end{bmatrix}.$$

Since only w_{s1} and w_{s3} are potential leaving variables, we need to look at only their components from this direction vector. Using the ratio test, we find that w_{s3} would be our leaving variable ($\lambda = \frac{7}{10}$), generating the new basis matrix

$$\hat{B} = \begin{bmatrix} 4 & 1 & 3 & 14 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

with basis $\{y_1, y_2, y_3, y_4, w_{s1}, w_{s4}\}$. Our reduced costs would now be

$$\hat{\bar{c}}^T = [0 \ 0 \ 0 \ 0]$$

$$\begin{aligned} & - [24 \ 24 \ 23 \ 89 \ 0 \ 0] \begin{bmatrix} 4 & 1 & 3 & 14 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}^{-1} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ & = - \left[\frac{11}{2} \ 2 \ 0 \ \frac{25}{2} \ \frac{5}{2} \ 0 \right] \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ & = \left[\frac{11}{2} \ 2 \ \frac{25}{2} \ \frac{5}{2} \right]. \end{aligned}$$

Since all reduced costs are positive (for our minimization problem), our current solution

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ w_{s1} \\ w_{s4} \end{bmatrix} = \begin{bmatrix} 4 & 1 & 3 & 14 & 0 & 0 \\ 1 & 3 & 2 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}^{-1} \begin{bmatrix} 13 \\ 5 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{4}{5} \\ 0 \\ 0 \\ \frac{7}{10} \\ \frac{4}{5} \\ \frac{7}{10} \end{bmatrix}$$

is the optimal solution to the dual problem. From our reduced cost calculations, we know that the optimal solution to the primal problem is $(\frac{11}{2}, 2, 0, \frac{25}{2}, \frac{5}{2}, 0)$.

Can we simplify any of this work for a general problem? We can see that some vectors repeated themselves in different context. For example, the dual reduced cost vector was exactly the value of the primal basic variables. We can exploit this, but only if we are careful.

In general, suppose we have a primal linear program in canonical form

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \\ & A \mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

with basis matrix B for which $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N) = (B^{-1}\mathbf{b}, \mathbf{0})$ is a basic solution but not feasible and its corresponding dual solution $\mathbf{y}^T = \mathbf{c}_B^T B^{-1}$ is feasible to the dual; thus, \mathbf{x} satisfies the optimality and complementary slackness conditions but not feasibility. The only way for \mathbf{x} to be primal infeasible is there to exist at least one basic variable $x_k < 0$. The dual problem is

$$\begin{aligned} & \min \quad \mathbf{b}^T \mathbf{y} \\ \text{s.t.} \\ & A^T \mathbf{y} \geq \mathbf{c}, \end{aligned}$$

which, when the matrix A is partitioned into B and N and surplus variables \mathbf{w} are added and similarly partitioned, we can rewrite as

$$\begin{aligned} & \min \quad \mathbf{b}^T \mathbf{y} + \mathbf{0}^T \mathbf{w} \\ \text{s.t.} \\ & \begin{bmatrix} B^T & -I & 0 \\ N^T & 0 & -I \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{w}_B \\ \mathbf{w}_N \end{bmatrix} \geq \begin{bmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{bmatrix} \\ & \mathbf{w}_B, \mathbf{w}_N \geq \mathbf{0}. \end{aligned}$$

By complementary slackness, we know that, at our current primal basic solution, $\mathbf{w}_B = \mathbf{0}$ and that $\mathbf{w}_N = -\bar{\mathbf{c}}_N$, the negative of the reduced cost vector for the primal nonbasic variables. The dual basic variables are \mathbf{y} and \mathbf{w}_N with matrices

$$\hat{B} = \begin{bmatrix} B^T & 0 \\ N^T & -I \end{bmatrix} \quad \hat{N} = \begin{bmatrix} -I \\ 0 \end{bmatrix}.$$

Because the variables \mathbf{y} are unrestricted in sign in the dual, and \hat{B} is a valid basis matrix, \mathbf{y} can always remain basic to the dual problem.

If we employ the simplex method on the dual problem, we obtain the following calculations (see Exercise 11.18):

1. The reduced cost vector $\hat{\mathbf{c}}_B$ associated with the dual nonbasic variables \mathbf{w}_B is

$$\hat{\mathbf{c}}_B = \mathbf{x}_B,$$

which are the values of the primal basic variables.

2. The components of the simplex direction \mathbf{d} corresponding to the basic dual variables w_N are

$$\mathbf{d}_{w_N}^T = \mathbf{e}_k^T B^{-1} N,$$

which is the k th row of the matrix $B^{-1}N$.

3. The ratio test becomes the calculation

$$\lambda_{\max} = \min \left\{ \frac{\bar{c}_j}{d_{w_j}} : d_{w_j} < 0 \right\},$$

where \bar{c}_j is the reduced cost of the primal nonbasic variable x_j and d_{w_j} is the component of the simplex direction vector \mathbf{d}_{w_N} corresponding to the dual basic variable w_j .

Each of these relevant simplex calculations can be made without explicitly writing down the dual problem! This introduces an algorithm to solve a linear program where we have a primal basic solution that corresponds, via complementary slackness, to a dual basic feasible solution; this algorithm, known as the **Dual Simplex Method**, solves the primal linear program by implicitly solving its dual problem, and is outlined in Algorithm 11.1.

Algorithm 11.1 Dual Simplex Method (Maximization Problem)

Step 0: Initialization. Start with a linear program in canonical form. Find an initial basis matrix B where the vector $\mathbf{y}^T = \mathbf{c}_B^T B^{-1}$ is a feasible solution to the dual problem. Calculate the primal reduced cost vector $\bar{\mathbf{c}}_N^T = \mathbf{c}_N^T - \mathbf{y}^T N$.

Step 1: Check Primal Feasibility. If $\mathbf{x}_B = B^{-1}\mathbf{b} \geq \mathbf{0}$, then STOP; $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N) = (B^{-1}\mathbf{b}, \mathbf{0})$ is an optimal basic feasible solution. Otherwise, choose a leaving variable x_k among the basic “variables x_j ” for which $x_j < 0$.

Step 2: Find Direction Vector. Calculate the vector $\mathbf{d}^T = \mathbf{e}_k^T B^{-1} N$ (the k th row of $B^{-1}N$).

Step 3: Perform Ratio Test. If all coordinates of \mathbf{d} are nonnegative, STOP; the dual linear program is unbounded, and so the primal linear program is infeasible. Otherwise, choose the primal entering value x_e by computing the maximum step size according to the ratio test

$$\lambda = \min \left\{ \frac{\bar{c}_j}{d_j} : d_j < 0 \right\}.$$

Step 4: Update Solution and Basis. Update B , N , \mathbf{x}_B , \mathbf{x}_N , \mathbf{y} , and $\bar{\mathbf{c}}_N$. Return to Step 1.

■ EXAMPLE 11.1

Let's illustrate the dual simplex method on our original problem (11.2). We have identified the basis matrix

$$B = \begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 3 & 2 & 0 & 0 \\ 14 & 6 & 0 & 1 \end{bmatrix},$$

which generates the primal basic solution $(5, 4, 0, 7, 0, -5)$ and the dual feasible solution $\mathbf{y} = \mathbf{y} = (\frac{11}{5}, 0, \frac{7}{5})$ and reduced cost vector $\bar{\mathbf{c}}_N = \left(-\frac{11}{5}, -\frac{7}{5}\right)$. Since $s_4 = -5 < 0$ in our current primal solution, s_4 will be our leaving variable; this variable corresponds to the 4th column of B . We next generate the 4th row of $B^{-1}N$ by

$$\begin{aligned} \mathbf{d}_4^T &= [0 \ 0 \ 0 \ 1] \begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 3 & 2 & 0 & 0 \\ 14 & 6 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \\ &= [-2 \ -2]. \end{aligned}$$

By our ratio test

$$\lambda_{\max} = \min \left\{ \frac{-\frac{11}{5}}{-2}, \frac{-\frac{7}{5}}{-2} \right\} = \frac{7}{10},$$

we have that s_3 is our entering variable, generating the basis matrix

$$B = \begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 3 & 1 & 0 \\ 3 & 2 & 0 & 1 \\ 14 & 6 & 0 & 0 \end{bmatrix}$$

and primal basic solution $\mathbf{x} = \left(\frac{11}{2}, 2, 0, \frac{25}{2}, \frac{5}{2}, 0\right)$. Since this is a feasible solution, it is an optimal solution to our primal linear program. The corresponding dual solution is $\mathbf{y} = \left(\frac{4}{5}, 0, 0, \frac{7}{10}, \frac{4}{5}, \frac{7}{10}\right)$.

The dual simplex method is not restricted to only those cases where we add constraints to the primal problem after we have initially solved it. As the next example demonstrates, as long as we have a primal basic solution that generates a dual feasible solution, we can use the dual simplex method to solve the problem.

EXAMPLE 11.2

Consider the following linear program:

$$\begin{aligned} \max \quad & (-3x_1 - 4x_2 - 2x_3) \\ \text{s.t.} \quad & 2x_1 - x_2 - 3x_3 + s_1 = -1 \\ & -x_1 - x_2 + 2x_3 + s_2 = -4 \\ & x_1, x_2, x_3, s_1, s_2 \geq 0. \end{aligned}$$

and its dual

$$\begin{aligned} \min \quad & (-y_1 - 4y_2) \\ \text{s.t.} \quad & 2y_1 - y_2 \geq -3 \\ & -y_1 - y_2 \geq -4 \\ & -3y_1 + 2y_2 \geq -2. \end{aligned}$$

If we choose s_1, s_2 as our initial primal basic variables, we have that $\mathbf{x}_B = (s_1, s_2) = (-1, -4)$ is not feasible, while the vector

$$\mathbf{y} = \mathbf{c}_B^T \mathbf{B}^{-1} = [0 \ 0]$$

is dual feasible with reduced costs $\bar{\mathbf{c}}_N = (\bar{c}_1, \bar{c}_2, \bar{c}_3) = (-3, -4, -2)$. Using the dual simplex method to solve the primal problem, we can let either s_1 or s_2 be the leaving variable; we shall choose s_2 using a variant of Dantzig's rule.

Next, to determine the entering variable, we calculate

$$\begin{aligned} \mathbf{d}^T &= [d_{x_1} \ d_{x_2} \ d_{x_3}] \\ &= [0 \ 1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 2 & -1 & -3 \\ -1 & -1 & 2 \end{bmatrix} \\ &= [-1 \ -1 \ 2]. \end{aligned}$$

The ratio test gives

$$\min \left\{ \frac{(-3)}{-1}, \frac{(-4)}{-1} \right\} = 3,$$

which implies that x_1 is the entering variable. Updating the solution yields $B = \{x_1, s_1\}$ and $N = \{x_2, x_3, s_2\}$ and basis matrix

$$B = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix},$$

giving the primal basic solution $\mathbf{x}_B = (4, -9)$ and dual solution $\mathbf{y}^T = \mathbf{c}_B^T \mathbf{B}^{-1} = (0, 3)$. Since $s_1 < 0$, this is our leaving variable. We next find our direction

$$\begin{aligned}
\mathbf{d}^T &= [d_{x_2} \quad d_{x_3} \quad d_{s_2}] \\
&= [0 \quad 1] \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -3 & 0 \\ -1 & 2 & 1 \end{bmatrix} \\
&= [-3 \quad 1 \quad 2],
\end{aligned}$$

while the ratio test indicates that x_2 must be entering, since it corresponds to the only negative component of \mathbf{d} . Our new basis is $\{x_1, x_2\}$ with basis matrix

$$B = \begin{bmatrix} 2 & -1 \\ -1 & -1 \end{bmatrix},$$

which generates the primal basic solution $\mathbf{x}_B = (1, 3)$. Since this solution is feasible, $(1, 3, 0, 0, 0)$ is our optimal solution.

The dual simplex method is an important algorithm both theoretically and computationally. We have already seen that it is useful when we have a basic solution to our primal problem and a dual feasible solution. This situation often occurs when solving integer programs, and we will return to this in Chapter 14. In addition, once a modification is made that finds an initial solution, dual simplex typically is the faster of the two simplex methods for solving linear programs. In fact, most software packages will use a dual simplex implementation to solve the given problems, when given a choice. However, we still need to be careful; because the dual simplex method is a specialized version of the simplex method, cycling can occur, so steps need to be taken to ensure convergence. For more information, check out the book by Bertsimas and Tsitsiklis [14].

11.2 TRANSPORTATION PROBLEM

In Section 2.9, we introduced the transportation problem as a special network model. In this problem, there are supply nodes that directly ship goods to demand nodes. In general, if there are m supply nodes, each with supplies s_i , and n demand nodes, each with demand d_j , and the cost associated with shipping items from supply node i to demand node j is c_{ij} , then the transportation problem is

$$\begin{aligned}
& \min \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^n x_{ij} \leq s_i, \quad i \in \{1, \dots, m\} \\
& \sum_{i=1}^m x_{ij} \geq d_j, \quad j \in \{1, \dots, n\} \\
(11.5) \quad & x_{ij} \geq 0, \quad i \in \{1, \dots, m\}, \quad j \in \{1, \dots, n\}.
\end{aligned}$$

The constraint matrix A has $m + n$ rows and mn columns, where the rows (constraints) correspond to each node and the columns (variables) correspond to each arc in the network; that is, the matrix A is the incidence matrix for the network. When solving this problem, we assume that

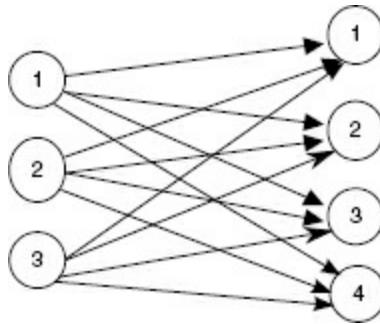
$$(11.6) \quad \sum_{i=1}^m s_i = \sum_{j=1}^n d_j,$$

that is, the total supply equals the total demand, which makes the constraints in (11.5) equalities. This is not a simplifying assumption since any transportation problem can be transformed into an equivalent problem with this property (see Exercises 2.44 and 2.45). Such problems that satisfy (11.6) are called *balanced transportation problems*. We can solve this using the simplex method, but careful analysis indicates that a more efficient version is possible.

Throughout this section, we will illustrate using the example

$$\begin{aligned}
& \min \quad 20x_{11} + 45x_{12} + 35x_{13} + 10x_{14} + 35x_{21} + 35x_{22} \\
(11.7) \quad & \quad + 50x_{23} + 20x_{24} + 30x_{31} + 20x_{32} + 15x_{33} + 25x_{34} \\
\text{s.t.} \quad & x_{11} + x_{12} + x_{13} + x_{14} = 50 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 30 \\
& x_{31} + x_{32} + x_{33} + x_{34} = 20 \\
& x_{11} + x_{21} + x_{31} = 30 \\
& x_{12} + x_{22} + x_{32} = 25 \\
& x_{13} + x_{23} + x_{33} = 40 \\
& x_{14} + x_{24} + x_{34} = 5 \\
& x_{ij} \geq 0, \quad i \in \{1, 2, 3\}; \quad j \in \{1, 2, 3, 4\}.
\end{aligned}$$

FIGURE 11.1 Transportation network.



Here there are $m = 3$ supply nodes and $n = 4$ demand nodes. Graphically, this problem is represented in [Figure 11.1](#).

If we consider (11.5) and we add all supply constraints and all demand constraints, the resulting two equalities would be the same. This implies that the rank of A is less than $m + n$. Before we actually determine the rank of A , we need a few definitions. Given a directed graph G , the underlying graph G_U is an undirected graph over the same set of nodes as G where (i, j) is an edge in G_U if and only if either (i, j) or (j, i) is a directed arc in G . In an undirected graph G_U , a subgraph T is a *spanning tree* if T is connected and contains no cycles. Finally, in a directed network G , a *spanning tree* on G is the set of arcs that correspond to a spanning tree on the underlying graph G_U of G .

■ EXAMPLE 11.3

Consider the transportation network given in [Figure 11.1](#). The arcs

$$T = \{(1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (3, 4)\}$$

correspond to a spanning tree on the network since, if we remove the arc orientations, the resulting undirected edges form a spanning tree.

Lemma 11.1 $\text{Rank}(A) = m + n - 1$. Furthermore, an $(m + n - 1, m + n - 1)$ sub-matrix B has full row rank if and only if the submatrix B corresponds to the node–arc matrix of a spanning tree of the network.

Proof See Exercise 11.7.

If we use the simplex method to solve (11.5), then by Lemma 11.1 we can eliminate any constraint from our original problem. In addition, any basic (feasible) solution to (11.5) corresponds to a spanning tree of the network.

■ EXAMPLE 11.4

In our example problem (11.7), consider the variables x_{11} , x_{22} , x_{23} , x_{32} , x_{33} , and x_{34} . If these were the basic variables of some basic solution, we'd solve the system of equations

$$\begin{array}{lll} x_{11} & & = 50 \\ x_{22} + x_{23} & & = 30 \\ & x_{32} + x_{33} + x_{34} & = 20 \\ x_{11} & & = 30 \\ x_{22} + x_{23} & & = 25 \\ x_{23} & + x_{33} & = 40 \\ & x_{34} & = 5. \end{array}$$

Even though we have the correct number of variables for a basic solution, there are no values for the variables that satisfy all these equations. If we eliminate the first constraint (remember, we can eliminate any of them), the basis matrix B would be

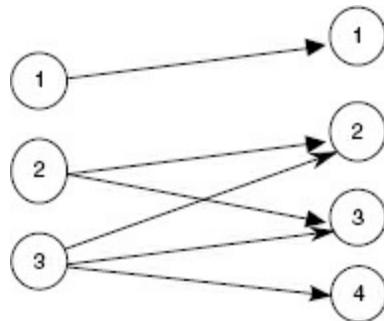
$$B = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This matrix has rank < 6 since the rows are not linearly independent:

$$0 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + 0 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The columns whose coefficients are not zero correspond to those arcs that make up a cycle. In fact, if we start the cycle at the supply node 2, then those arcs with $+1$ coefficients are moving in the “correct” direction of the arc, and those arcs with -1 coefficients are moving in the “opposite” direction of the arc. The cycle in this case could be $x_{22} \rightarrow x_{23} \rightarrow x_{33} \rightarrow x_{32} \rightarrow x_{22}$, which is seen in [Figure 11.2](#).

FIGURE 11.2 “Solution” with a cycle.



Solving the Transportation Problem: Transportation Simplex

Since the transportation problem is a linear program, we can employ the simplex method to solve (11.5). Knowing the structure of a basic (feasible) solution allows us to streamline the algorithm by taking advantage of the following characteristics:

1. When we find an entering variable, it is equivalent to adding an arc to our spanning tree. This creates a unique cycle in our network.
2. When we find a leaving variable, we are essentially removing an arc to generate a new spanning tree. The arc that is removed must be on the unique cycle.
3. We want our entering variable to have positive value at the new solution; hence, in order to keep the constraints satisfied, we must add and subtract amounts from arcs in the cycle until one of them becomes zero.
4. We can use duality to find our reduced costs. This is because each variable appears in exactly two constraints. Hence, each dual constraint has only two variables.

We now consider an alternative version of the simplex method, called appropriately enough the **Transportation Simplex Method**, that exploits these characteristics to improve the efficiency of our algorithm. Before we begin, it will be useful to rewrite our problem into a more readable (and workable) form. Typically, transportation problems are written in a tabular form, where the rows correspond to the supply constraints, the columns correspond to the demand constraints, and each cell is a variable x_{ij} . Within each cell, in the upper left corner, is the cost associated with variable x_{ij} . The last column of the tableau gives the supply values s_i , while the bottom row

gives the demand values d_j . A nice feature of this form is that, in order to check whether a solution is feasible, we only need to sum across a row and down a column. For our example problem, the tabular form of the transportation problem is as follows:

20	45	35	10		
x_{11}	x_{12}	x_{13}	x_{14}	50	
35	35	50	20		
x_{21}	x_{22}	x_{23}	x_{24}	30	
30	20	15	25		
x_{31}	x_{32}	x_{33}	x_{34}	20	
30	25	40	5		

(11.8)

If we are to use this tabular form, we must find out what a basic feasible solution looks like. Recall that the basic variables associated with a basic (feasible) solution form a spanning tree in the network form of the problem. For example, in (11.7), the solution $x_{11} = 30, x_{13} = 15, x_{14} = 5, x_{22} = 25, x_{23} = 5, x_{33} = 20$ is a basic feasible solution, since the graph of the basic variables above is a spanning tree. In the tabular form, these variables do not yield a “cycle” either. This idea will help identify a basic (feasible) solution in the tabular form.

Cycle in Tableau An ordered sequence of at least four different cells/variables is a *cycle* if each of the following conditions hold:

1. Any two consecutive cells lie in either the same row or in the same column.
2. No three consecutive cells lie in the same row or column.
3. The last cell in the sequence has a row or column in common with the first cell in the sequence.

■ EXAMPLE 11.5

Using the tabular form given in (11.8), the following sequences are cycles:

1. $x_{11} \rightarrow x_{13} \rightarrow x_{33} \rightarrow x_{31}$.
2. $x_{12} \rightarrow x_{32} \rightarrow x_{33} \rightarrow x_{23} \rightarrow x_{24} \rightarrow x_{14}$.
3. $x_{11} \rightarrow x_{12} \rightarrow x_{32} \rightarrow x_{34} \rightarrow x_{24} \rightarrow x_{21}$.

The following ordered sequences are not cycles:

$$1. x_{11} \rightarrow x_{12} \rightarrow x_{13} \rightarrow x_{33} \rightarrow x_{32} \rightarrow x_{31}.$$

$$2. x_{11} \rightarrow x_{12} \rightarrow x_{24} \rightarrow x_{34} \rightarrow x_{33} \rightarrow x_{31}.$$

A tabular form of a cycle seems very similar to the graph-theoretic version of a cycle. In fact, these are tied together quite nicely.

In a balanced transportation problem with m supply points and n demand points, the cells corresponding to a set of $m + n - 1$ variables contain no cycles if and only if the subgraph in the network corresponding to these variables contains no cycles.

In a balanced transportation problem with m supply points and n demand points, the cells corresponding to a set of $m + n - 1$ variables contain no cycles if and only if the $m + n - 1$ variables yield a basic solution.

Thus, in each iteration of our algorithm, if we maintain a feasible solution of $m + n - 1$ variables that does not contain a cycle in the tabular form, then the solution is a basic feasible solution, and hence could be a solution obtained by the simplex method. In the remainder of this section, we use this structure to (a) find an initial basic feasible solution very quickly; (b) determine whether our current solution is optimal, and if not, find an entering variable to make basic; and (c) determine how to find our entering variable and update our basic feasible solution.

Determining an Initial Basic Feasible Solution Why not just solve the Phase I problem to find our initial basic feasible solution. To do this, we would have to add $m + n$ artificial variables, which would take at least that many iterations of the simplex method to remove them from the basis. By using the special structure of the balanced transportation problem, we can design an efficient heuristic method for finding our initial solution.

The **minimum cost method** discussed here is a very simple algorithm for finding a starting basic feasible solution. Its approach is to find a cell/variable with low cost and assign as much as possible to it. By doing this, we satisfy at least one constraint (and perhaps two), and so we only need to worry about satisfying the remainder. More formally, the minimum cost method is given in Algorithm 11.2.

Algorithm 11.2 Minimum Cost Method

Step 0: Assume that all cells are empty and not crossed out.

Step 1: From those cells not crossed out, choose the cell x_{ij} with smallest cost c_{ij} .
Step 2: Set $x_{ij} \leftarrow \min\{s_i, d_j\}$, its largest possible value (0 is a possible value).
Step 3: Reduce s_i and d_j each by x_{ij} .
Step 4: Cross out exactly one of the row i or column j whose s_i or d_j is 0.
Step 5: Repeat steps 1–5 until only one cell can be chosen. At this point, stop.

■ EXAMPLE 11.6

Let's use the minimum cost method to find an initial solution to (11.7). Initially, we would have the tableau

20	45	35	10		50
35	35	50	20		30
30	20	15	25		20
30	25	40	5		

Since cell x_{14} has the smallest cost ($c_{14} = 10$), we put $5 = \min\{5, 50\}$ into that cell. The supply of row 1 is reduced by 5 to 45, while the demand of column 4 is reduced to 0; hence, we cross out column 4, since assigning any value to these cells would make our solution infeasible, and get the tableau

20	45	35	10	5	45
35	35	50	20	X	30
30	20	15	25	X	20
30	25	40	0		

We next choose cell x_{33} with cost 15, and assign $20 = \min\{20, 40\}$ to that cell. The supply of row 3 becomes 0 while the demand of column 3 becomes 20. We cross out cells x_{31} and x_{32} , giving the tableau

20	45	35	10	5	45
35	35	50	20	X	30
30	20	15	25	X	0
X	X		20		
30	25	20	0		

Next, we choose cell x_{11} , assign $30 = \min\{30, 45\}$ to it, and cross out cell x_{21} . The next tableau is

20	45	35	10		
30				5	15
35	35	50	20		
X				X	30
30	20	15	25		
X	X	20		X	0
0	25	20	0		

At this point, we have a choice between cells x_{13} and x_{22} . It does not matter which we choose, so we randomly select x_{22} . Assigning $25 = \min\{30, 25\}$ to this cell reduces the supply of row 2 to 5 and the demand of column 2 to 0. This gives the tableau

20	45	35	10		
30		X		5	15
35	35	50	20		
X		25		X	5
30	20	15	25		
X	X	20		X	0
0	0	20	0		

Our next cell is x_{13} , where we put 15 units. Note that we do not have any cells to cross out in row 1. Our updated tableau is now

20	45	35	10		
30		X	15	5	0
35	35	50	20		
X		25		X	5
30	20	15	25		
X	X	20		X	0
0	0	5	0		

Finally, our last cell satisfies the last two remaining constraints, giving the basic feasible solution

20	45	35	10		
30			15	5	50
35	35	50	20		
		25	5		30
30	20	15	25		
		20			20
30	25	40	5		

At each step of the minimum cost method, we satisfy at least one constraint, and at the last step we satisfy two. Since there are $m + n$ constraints, we are left with at most $m + n - 1$ filled cells. Furthermore, because we cross out unavailable cells throughout, we are guaranteed to find a basic feasible solution to(11.7).

It is possible that, when the minimum cost method is finished, there are less

than $m + n - 1$ filled cells. This indicates that there is a degenerate basic feasible solution, and so we need to determine the remaining basic variables. In this case, we can set cells to 0 so long as they do not force a cycle in our tableau, until we have $m + n - 1$ filled cells (see Exercise 11.8).

Determining Optimality: Checking Reduced Costs In the simplex method, we would check for any negative reduced cost (remember, minimization problem), and if there is none then we are at the optimal solution. How can we check reduced costs in this array form? We start with linear programming duality. For a general balanced transportation problem (11.5), if we associate the dual variables u_i for each supply constraint and v_j for each demand constraint, we'd have the following dual linear program:

$$\begin{aligned} \max \quad & \sum_{i=1}^m s_i u_i + \sum_{j=1}^n d_j v_j \\ \text{s.t.} \quad & u_i + v_j \leq c_{ij}, \quad i \in \{1, 2, \dots, m\}, \quad j \in \{1, 2, \dots, n\} \\ & u_i, v_j \text{ unrestricted in sign.} \end{aligned}$$

We know from duality theory that given a basic feasible solution to a primal linear program, those constraints in the dual corresponding to the basic variables are satisfied at equality. Note that there are $m + n$ variables in the dual, while the number of basic variables in the primal (and hence, equality constraints to be solved in the dual) is $m + n - 1$. We determine values for these $m + n$ dual variables by fixing one of them to some value and solving for the remainder. We then compute our reduced costs

$$\bar{c}_{ij} = c_{ij} - (u_i + v_j).$$

If every \bar{c}_{ij} is nonnegative then our current solution is optimal, otherwise any variable with negative \bar{c}_{ij} is eligible to enter the basis.

■ EXAMPLE 11.7

Let's see if the initial basic feasible solution found in Example 11.6 is optimal. From the basic variables $x_{11}, x_{13}, x_{14}, x_{22}, x_{23}, x_{33}$, we have the following dual constraints that must be satisfied at equality:

$$\begin{array}{rclcl}
u_1 & + v_1 & = 20 & & (x_{11}) \\
u_1 & + v_3 & = 35 & & (x_{13}) \\
u_1 & + v_4 & = 10 & & (x_{14}) \\
u_2 & + v_2 & = 35 & & (x_{22}) \\
u_2 & + v_3 & = 50 & & (x_{23}) \\
u_3 & + v_3 & = 15 & & (x_{33}).
\end{array}$$

Since there are six constraints and seven variables, if we fix one of the variables, then we can uniquely solve for the remaining ones, since the rank of this matrix is 6. It is often easiest to set one of the variables to 0; here, we will set $v_1 = 0$, which leads to the solution

$$u_1 = 20, u_2 = 35, u_3 = 0, v_1 = 0, v_2 = 0, v_3 = 15, v_4 = -10.$$

This is often represented as an addition row and column in the tableau:

$u_i \setminus v_j$	0	0	15	-10	
20	20 30	45 35	35 25	10 20 5	50 30
35	35 25	35 20	50 15 20	20 25 5	
0	30 20	20 15 20	15 20	25 5	20
	30	25	40	5	

For each of the remaining cells, we calculate $\bar{c}_{ij} = c_{ij} - (u_i + v_j)$. If $\bar{c}_{ij} < 0$, we are not at an optimal solution, since we are minimizing our objective function. Below is the calculation of the reduced costs for each nonbasic cell; these values are circled in the given cells.

$u_i \setminus v_j$	0	0	15	-10	
20	20 30	45 35 25	35 15	10 5 -5	50 30 -5
35	35 0	35 25	50 5	20 -5	
0	30 30	20 20	15 20	25 35	20
	30	25	40	5	

Note that the reduced cost for cell x_{24} is negative, indicating that our current solution is not optimal. However, we can improve our solution by letting variable x_{24} enter the basis.

Updating the Basic Feasible Solution Once we know our current solution is

not optimal and have identified an entering variable, we need to find a leaving variable and then update our solution. Recall that, when we add our entering variable as an arc in the network, we get a cycle. This cycle corresponds to a cycle in the tableau. In fact, in each column or row, there will be either 0 or 2 cells from the cycle in that column or row. Consider the row i containing the entering variable/cell x_{ij} . There must be a variable x_{ij_1} from our cycle in this row. If our entering variable is to increase from zero, x_{ij_1} must decrease its value by the same amount, in order to maintain feasibility. Now consider the column j_1 containing this cell x_{ij_1} . There must be another cell x_{i,j_1} in this column that is also in our cycle. In order to maintain feasibility, this cell must increase its value. We continue this until we get to a cell in the column containing our entering cell. Since there is no explicit upper bound to any variable, the maximum value that our entering variable can have must be the minimum value of all those cells that must decrease in value to maintain feasibility. We formalize this idea below.

To find the leaving variable and update the solution:

1. Find the cycle containing the entering variable and some of the basic variables.
2. Counting only those variables in the cycle, label those from Step 1 that are an even number of cells away from the entering variable (including the entering variable itself) as (+) cells, and those that are an odd number of cells away from the entering variable as (-) cells.
3. Among all (-) variables, find the variable(s) that has the smallest value. Let θ denote this value. For each (+) variable, and the entering variable, increase its value by θ . For each (-) variable, decrease its value by θ . Among all the variables with initial value θ , choose one as the leaving variable. All others will remain basic, even if they have value 0.

■ EXAMPLE 11.8

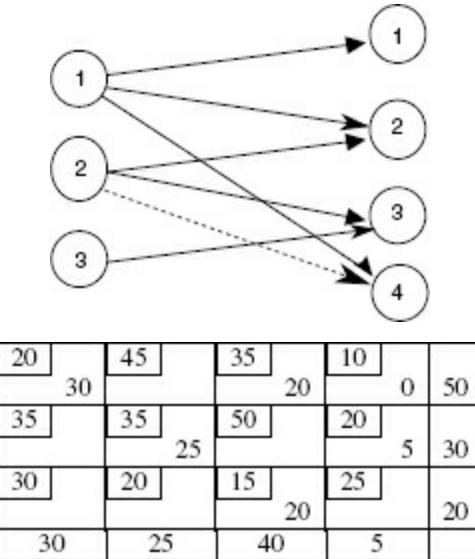
In Example 11.7, we found that variable x_{24} was the entering variable. Adding this variable into the tableau gives the cycle $x_{24} \rightarrow x_{23} \rightarrow x_{13} \rightarrow x_{14} \rightarrow x_{24}$. We have variables x_{24} and x_{13} as (+) variables, while x_{23} and x_{14} as (-)

variables. Below is the tableau as formed:

$u_i \setminus v_j$	0	0	15	-10	
20	20 30	45	35 15	10 5	50
35	35 25	35 20	50 5	20 0	30
0	30 20	20 15	15 20	25	20
	30	25	40	5	

Since the smallest value in the (-) cells is $\theta = 5$, we increase variables x_{24} and x_{13} by 5 and decrease x_{23} and x_{14} by 5. Note that, since each of x_{23} and x_{14} has initial value θ , we can choose either as the leaving variable; we shall choose x_{23} as our leaving variable. This gives us a degenerate basic feasible solution. Our updated basic feasible solution would then be

FIGURE 11.3 Network representation of basic feasible solution and entering variable.



Our new basic variables are $x_{11}, x_{13}, x_{14}, x_{22}, x_{24}, x_{33}$.

Now consider the above example from a network perspective. In [Figure 11.3](#), the original basic feasible solution are the arcs in our network, with the dashed line representing the entering variable x_{24} .

Note that we have the cycle $x_{24} \rightarrow x_{14} \rightarrow x_{12} \rightarrow x_{22}$ that corresponds to the cycle found in the tableau. Furthermore, those (+) cells correspond to arcs in

the network cycle that are in the correct orientation of the cycle, and (-) cells correspond to arcs in the network cycle that are in the opposite orientation of the cycle.

Once we add our entering variable, our cycles can be much more elaborate.

■ EXAMPLE 11.9

Suppose that we are given the following tableau, with appropriate basic feasible solution:

20	45	35	10		25
35					
35	35	50	20		50
10	20	20			
30	20	15	25		
		10	30	30	40
45	20	30	30		

If x_{14} is to be the entering variable, our cycle would be $x_{14} \rightarrow x_{34} \rightarrow x_{33} \rightarrow x_{23} \rightarrow x_{12} \rightarrow x_{11} \rightarrow x_{14}$. Our tableau, with appropriately labeled basic variable cells, is

20	-	45	35	10	+	θ	25
35							
35	+	35	50	-	20		50
10		20	20				
30	20	15	25	-	30	30	40
		10					
45	20	30	30				

Setting $\theta = \min\{30, 20, 35\} = 20$, our updated basic feasible solution would be

20	45	35	10	20	25
15					
35	35	50	20		50
30	20				
30	20	15	25	10	40
		30			
45	20	30	30		

Putting this all together gives us the entire transportation simplex method given in Algorithm 11.3.

Algorithm 11.3 Transportation Simplex Method

Step 1: If the problem is unbalanced, balance it.

Step 2: Use the minimum cost method (or some other algorithm) to find an initial basic feasible

solution.

Step 3: Use the fact that $c_{ij} = u_i + v_j$ for all basic variables, and that one variable (say u_1 can be fixed to 0), to find all dual variables for the current basic feasible solution.

Step 4: If $c_{ij} - u_i - v_j \geq 0$ for all nonbasic variables, STOP. We are at an optimal solution. Otherwise, choose an entering variable among those nonbasic variables where $c_{ij} - u_i - v_j < 0$.

Step 5: Using the unique cycle formed by adding the entering variable, identify the leaving variable and update the basic feasible solution.

Step 6: Using the new basic feasible solution, return to Step 3.

■ EXAMPLE 11.10

If we continue with Example 11.8, we now need to find the dual variable values and use them to calculate the reduced cost for each nonbasic variable. It should be easy to see that these values are given in the tableau

$u_i \setminus v_j$	0	5	15	-10	
20	20	45	35	10	
30	30	(20)	20	0	50
30	35	35	50	20	
0	(5)	25	(5)	5	30
0	30	20	15	25	
	30	25	40	5	

Since all reduced costs are positive, our current solution is optimal.

When updating the basic feasible solution, all we did was add and subtract values. This was much easier than using the general simplex method. Did you also notice that if our initial basic feasible solution was integral, then the next basic feasible solution was integral. This important property is formalized in the following theorem.

In a balanced transportation problem, if all supplies and demands are integers, then there exists an optimal solution where all variables have integer values. In addition, the transportation simplex method will find such a solution.

Because all our operations in the transportation simplex method involve comparisons (finding minima), addition, and subtraction, this is a very efficient implementation of the simplex method, and will solve the

transportation problem much more quickly compared to our simplex method. We will see a generalization of this approach for minimum cost network flows in Chapter 12.

11.3 COLUMN GENERATION

In many applications, there are a huge number of possible decisions, which translates to a huge number of columns in our linear program. One example of this occurred in Chapter 4 when we examined the crew scheduling problem for the airline industry, where there could easily be over 100 million possible combinations. At some point in the simplex method, we would have to find the reduced costs for nearly every variable. This frightening computational task led people to come up with ways to deal with this. One approach, known as **column generation** (or **generalized linear programming**), generates only columns as they are needed in the simplex method. A class of problems for which this technique is useful is the *Cutting Stock Problem*, which is illustrated by the following problem.

Problem: Materials such as paper, plastic film, and fabric are often manufactured in rolls of large width. These rolls, referred to as *raws*, are later cut into rolls of smaller widths, called *finals*. Typically, raws are produced in a few predetermined widths provided by the customer and which can vary greatly. Because of this, the manufacturer produces such widths by cutting the raws with a knife. For example, a raw of width 150 inches can be cut into three finals of width 30 inches and two finals of width 25 inches, leaving 10 inches of scrap material, or it can be cut into four finals of width 25 inches and three finals of width 15 inches, with 5 inches of waste. Given a list of orders from customers, the manufacturer would like to fill these orders from the existing raws so as to reduce the amount of waste material.

Let us consider an example where the raws are 100 inches wide and we have to fill the following summary of orders:

- 84 finals of width 38 inches.
- 60 finals of width 32 inches.
- 30 finals of width 27 inches.
- 20 finals of width 18 inches.

To use linear programming for this problem, suppose we let variable x_j correspond to a cutting pattern P_j , so that the column

$$\mathbf{a}_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ a_{3j} \\ a_{4j} \end{bmatrix}$$

corresponds to a_{1j} finals of width 38 inches, a_{2j} corresponds to finals of width 32 inches, a_{3j} corresponds to finals of width 27 inches, and a_{4j} corresponds to finals of width 18 inches. For example, the cutting pattern of two finals of width 32 inches and two finals of width 18 inches and have no waste correspond to the column

$$\mathbf{a}_1 = \begin{bmatrix} 0 \\ 2 \\ 2 \\ 0 \end{bmatrix}.$$

Variable x_j tells us how many raws should be cut using this pattern. Our constraints would be

$$\begin{aligned} \sum_{j=1}^n a_{1j}x_j &= 84 \\ \sum_{j=1}^n a_{2j}x_j &= 60 \\ \sum_{j=1}^n a_{3j}x_j &= 30 \\ \sum_{j=1}^n a_{4j}x_j &= 20, \end{aligned}$$

where n is the total number of patterns we have (which is unknown at this time). We will minimize the amount of scrap, which is calculated by

$$\text{Scrap} = 100 \sum_{i=1}^n x_i - [84(38) + 60(32) + 30(27) + 20(18)].$$

Since minimizing scrap is equivalent to

$$\min \sum_{i=1}^n x_i \text{ (why?)},$$

we will use this as our objective function.

In general, if we have a cutting stock problem where the raws are r inches wide and the order summary calls for b_i finals of width w_i ($i = 1, 2, \dots, m$), we

have the following linear program:

$$\begin{aligned} \min \quad & \sum_{i=1}^n x_i \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0. \end{aligned}$$

Each column $\mathbf{a} = (a_1, a_2, \dots, a_m)$ of A is a specific pattern for cutting the raw; that is, \mathbf{a} is a column of A if and only if $a_k \geq 0$ and integral for all k and

$$\sum_{k=1}^m w_k a_k \leq r.$$

Recall that at this point we do not know every column of A , only a small subset of them. Suppose that, for our reduced problem, we have solved the problem using the simplex method and we have the basis matrix B corresponding to the optimal basic variables. To check if this solution is optimal, we must compute the reduced costs for each nonbasic variable. But how can we do this if we don't know every column of A ? If we use the simplex method, we can first solve the following equation for \mathbf{y} :

$$\mathbf{y}^T B = \mathbf{c}_B^T = [1 \ 1 \ \dots \ 1].$$

Since all objective coefficients have value 1, when we examine the reduced costs, we are searching for a column \mathbf{a}_k of N such that

$$\bar{c}_k = 1 - \mathbf{y}^T \mathbf{a}_k < 0.$$

If no such column exists, we are at the optimal solution. In other words, given the vector \mathbf{y} , widths w_k and raw size r , we are looking for a vector \mathbf{a} of nonnegative integer components that satisfies the following conditions:

$$\begin{aligned} \sum_{j=1}^m y_j a_j &> 1 \\ \sum_{j=1}^m w_j a_j &\leq r. \end{aligned}$$

This can be done by solving the knapsack problem

$$\begin{aligned}
z^* &= \max \sum_{j=1}^m y_j a_j \\
\text{s.t.} \\
\sum_{j=1}^m w_j a_j &\leq r \\
a_j &\geq 0, \text{ integer}, \quad j \in \{1, \dots, m\}.
\end{aligned}$$

If $z^* > 1$, we have generated the column $\mathbf{a}_k = \mathbf{a}$ of our entering variable; if $z^* \leq 1$, there is no column that can enter the basis that would improve upon our current solution, so this solution is optimal.

■ EXAMPLE 11.11

Let's return to our original problem. To start, we need to obtain a set of columns that (a) are a possible cutting pattern, and (b) form a set of $m = 4$ linearly independent columns (so that we can obtain our first basis matrix B). For any problem, the first set of columns can be generated by m cutting patterns such that the i th pattern yields only finals of width w_i . In our case, we would have

$$B = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix},$$

which, along with $\mathbf{b} = (84, 60, 30, 20)$, yields $\mathbf{x}_B = (42, 20, 10, 4)$. Using the simplex method, we first solve the system of equations

$$\mathbf{y}^T B = [1 \ 1 \ 1 \ 1],$$

giving $\mathbf{y} = \left(\frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{5}\right)$. We next need to find a column \mathbf{a} that satisfies

$$38a_1 + 32a_2 + 27a_3 + 18a_4 \leq 100$$

$$\frac{1}{2}a_1 + \frac{1}{3}a_2 + \frac{1}{3}a_3 + \frac{1}{5}a_4 > 1.$$

This is done by solving the following knapsack problem:

$$\begin{aligned}
z^* &= \max \quad \frac{1}{2}a_1 + \frac{1}{3}a_2 + \frac{1}{3}a_3 + \frac{1}{5}a_4 \\
\text{s.t.} \\
38a_1 + 32a_2 + 27a_3 + 18a_4 &\leq 100 \\
a_k &\geq 0, \text{ integer}.
\end{aligned}$$

The optimal value is $z^* = \frac{6}{5}$ with the optimal vector $\mathbf{a} = (2, 0, 0, 1)$, which is our entering column. We then complete the simplex method by solving

$$B\mathbf{d} = -\mathbf{a},$$

giving $\mathbf{d} = (-1, 0, 0, -\frac{1}{5})$. Computing

$$\lambda = \min \left\{ \frac{42}{1}, \frac{4}{\left(\frac{1}{5}\right)} \right\} = 20,$$

indicating that the fourth column must leave the basis. Our basis matrix B is now

$$B = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Beginning our second iteration, we solve the system of equations

$$\mathbf{y}^T B = [1 \ 1 \ 1 \ 1],$$

giving $\mathbf{y} = \left(\frac{1}{2}, \frac{1}{3}, \frac{1}{3}, 0\right)$. This gives the knapsack problem

$$\begin{aligned} z^* = \max \quad & \frac{1}{2}a_1 + \frac{1}{3}a_2 + \frac{1}{3}a_3 + 0a_4 \\ \text{s.t.} \quad & 38a_1 + 32a_2 + 27a_3 + 18a_4 \leq 100 \\ & a_k \geq 0, \text{ integer,} \end{aligned}$$

whose solution is $\mathbf{a} = (1, 0, 2, 0)$ with value $z^* = \frac{7}{6}$; this is our entering column. To find our direction \mathbf{d} , we solve $B\mathbf{d} = -\mathbf{a}$, giving $\mathbf{d} = (-\frac{1}{2}, 0, -\frac{2}{3}, 0)$. Using $\mathbf{x}_B = B^{-1}\mathbf{b} = (22, 20, 10, 20)$, we can determine our leaving variable using

$$\lambda = \min \left\{ \frac{22}{\left(\frac{1}{2}\right)}, \frac{10}{\left(\frac{2}{3}\right)} \right\} = 15;$$

this implies the third column must leave the basis. Our basis matrix B is now

$$B = \begin{bmatrix} 2 & 0 & 1 & 2 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In our third iteration, we again solve

$$\mathbf{y}^T B = [1 \ 1 \ 1 \ 1],$$

giving $\mathbf{y} = \left(\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, 0\right)$. We next solve

$$\begin{aligned} z^* &= \max \quad \frac{1}{2}a_1 + \frac{1}{3}a_2 + \frac{1}{4}a_3 + 0a_4 \\ \text{s.t.} \end{aligned}$$

$$38a_1 + 32a_2 + 27a_3 + 18a_4 \leq 100$$

$$a_k \geq 0, \text{ integer,}$$

whose solution is $\mathbf{a} = (1, 1, 1, 0)$ with $z^* = \frac{13}{12}$, which gives our entering column. To find our direction \mathbf{d} , we solve $B\mathbf{d} = -\mathbf{a}$, giving $\mathbf{d} = (-\frac{1}{4}, -\frac{1}{3}, -\frac{1}{2}, 0)$. Using $\mathbf{x}_B = B^{-1}\mathbf{b} = (\frac{29}{2}, 20, 15, 20)$, we can determine our leaving variable using

$$\lambda = \min \left\{ \frac{\left(\frac{29}{2}\right)}{\left(\frac{1}{4}\right)}, \frac{20}{\left(\frac{1}{3}\right)}, \frac{15}{\left(\frac{1}{2}\right)} \right\} = 30;$$

this implies the third column must leave the basis. Our basis matrix B is now

$$B = \begin{bmatrix} 2 & 0 & 1 & 2 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In our fourth iteration, we again solve

$$\mathbf{y}^T B = [1 \ 1 \ 1 \ 1],$$

giving $\mathbf{y} = \left(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}, 0\right)$. The knapsack problem

$$\begin{aligned} z^* &= \max \quad \frac{1}{2}a_1 + \frac{1}{3}a_2 + \frac{1}{6}a_3 + 0a_4 \\ \text{s.t.} \end{aligned}$$

$$38a_1 + 32a_2 + 27a_3 + 18a_4 \leq 100$$

$$a_k \geq 0, \text{ integer}$$

gives the solution $\mathbf{a} = (2, 0, 0, 0)$ with $z^* = 1$. Hence, we have the optimal basis. Solving for \mathbf{x}_B gives $(7, 10, 30, 20)$, implying that we produce 7 raws cut by pattern $(2, 0, 0, 0)$, 10 raws of pattern $(0, 3, 0, 0)$, 30 raws of pattern $(1, 1, 1, 0)$, and 20 raws of pattern $(2, 0, 0, 1)$, totaling 67 raws.

Note that the final solution was integral. What is interesting about the cutting stock problem is that quite often, for no theoretical reason, the solution is integral. Of course, this does not always happen. However, when it does not happen, typically there is an integer solution with the same optimal value. However, getting such a solution is often difficult.

Cutting stock problems are not the only applications of column generation. For example, Degraeve and Schrage [28] use column generation to determine the production scheduling system for the curing operations at Bridgestone/Firestone Off-The-Road, which manufactures large tires for off-road machines. In this case, there are multiple constraints that determine a feasible tire mold formation, not just the one constraint we used in our example. Using real data from the manufacturing plant, less than 1000 of the millions of possible columns were generated during the algorithm.

11.4 DANTZIG–WOLFE DECOMPOSITION

In certain applications of linear programming, the constraints of the problem can be grouped into two classes: “easy” constraints and “hard” constraints. For example, the multicommodity flow problems we saw in Chapters 2 and 4 are such a case where the “hard” constraints correspond to the bounds on the total amount of flow on each arc and the “easy” constraints correspond to the flow constraints for each commodity. In general, the names “hard” and “easy” are misnomers; typically the “hard” constraints are the complicating ones, while the “easy” constraints would be those that, without the “hard” ones in the problem, would allow the problem to be efficiently solved.

When these applications have large numbers of variables and constraints, it is often useful to decompose our problem into one where we work on, at any given time, only the “easy” constraints or only the “hard” constraints. In this section, we discuss an approach for solving linear programs through such a decomposition method, known as **Dantzig–Wolfe Decomposition**. It is a form of column generation, where the columns are generated by solving the “easy” constraints along with some objective function.

Throughout this section, we will assume that our linear program can be

written in the form

$$\begin{aligned}
 & \max \quad \mathbf{c}^T \mathbf{x} \\
 & \text{s.t.} \\
 & \quad A_H \mathbf{x} = \mathbf{b}_H \\
 & \quad A_E \mathbf{x} = \mathbf{b}_E \\
 (11.9) \quad & \quad \mathbf{x} \geq \mathbf{0},
 \end{aligned}$$

where A_H (A_E) is the constraint matrix for the hard (easy) constraints.

■ EXAMPLE 11.12

We shall illustrate this approach with the following example problem throughout this section:

$$\begin{aligned}
 & \max \quad 8x_1 + 9x_2 + 7x_3 + 10x_4 \\
 & \text{s.t.} \\
 & \quad 2x_1 + x_2 + 4x_3 + 3x_4 = 20 \\
 & \quad x_1 + x_2 + x_3 + x_4 = 10 \\
 & \quad 3x_1 + x_2 \leq 6 \\
 & \quad 2x_3 + x_4 \leq 8 \\
 & \quad x_3 + x_4 \leq 6 \\
 (11.10) \quad & \quad x_1, x_2, x_3, x_4 \geq 0.
 \end{aligned}$$

The last three constraints will be considered the easy constraints and the first two the hard constraints, since if we remove the first two constraints we get the decoupled problems

$$\begin{aligned}
 & \max \quad 8x_1 + 9x_2 \\
 & \text{s.t.} \\
 & \quad 3x_1 + x_2 \leq 6 \\
 & \quad x_1, x_2 \geq 0
 \end{aligned}$$

and

$$\begin{aligned}
 & \max \quad 7x_3 + 10x_4 \\
 & \text{s.t.} \\
 & \quad 2x_3 + x_4 \leq 8 \\
 & \quad x_3 + x_4 \leq 6 \\
 & \quad x_3, x_4 \geq 0,
 \end{aligned}$$

which can both be easily solved using graphical methods ([Figures 11.4](#) and [11.5](#)). After adding slack variables to each constraint, we would have

FIGURE 11.4 Feasible region for (x_1, x_2) variables.

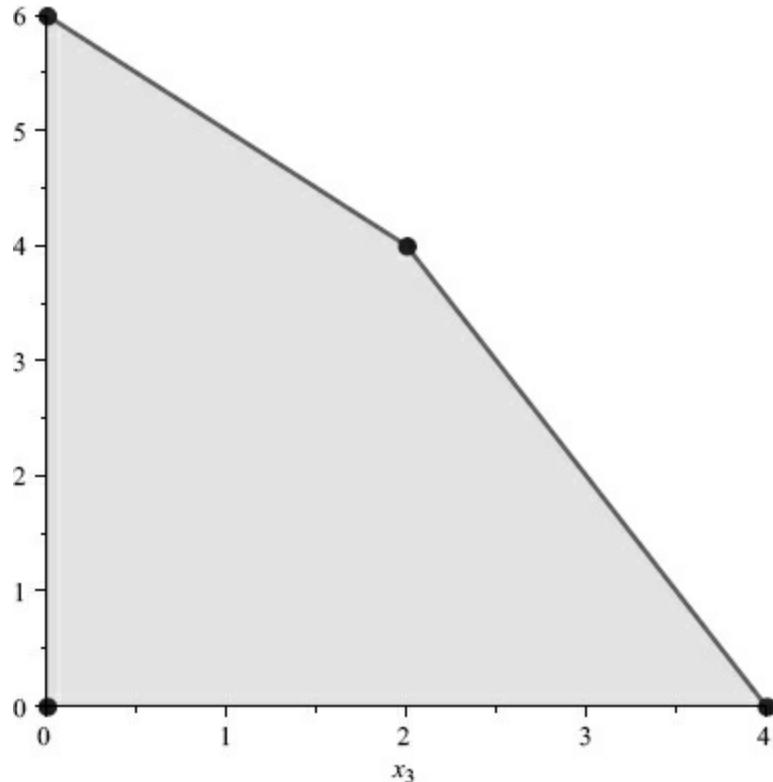
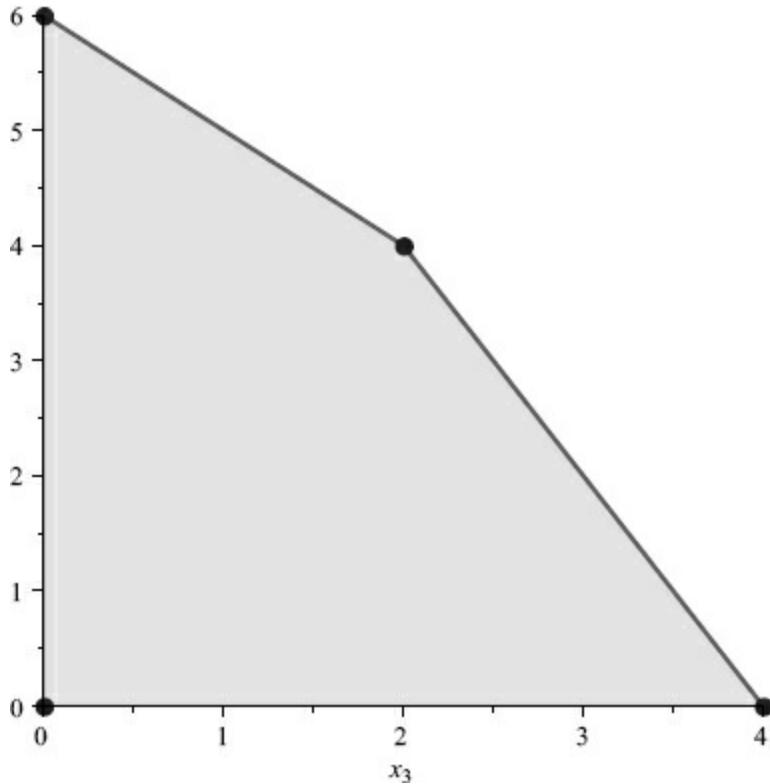


FIGURE 11.5 Feasible region for (x_3, x_4) variables.



$$A_H = \begin{bmatrix} 2 & 1 & 4 & 3 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$A_E = \begin{bmatrix} 3 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Consider the feasible region for the “easy” constraints

$$(11.11) X = \{x : A_E x = b_E, x \geq 0\}$$

and assume that X is bounded (this condition can be relaxed). By Theorem 7.4, if we have the set $V = \{v_1, v_2, \dots, v_K\}$ of extreme points of X , then every solution $x \in X$ can be written as a convex combination of the extreme points in V , that is,

$$x = \sum_{j=1}^K \alpha_j v_j$$

$$1 = \sum_{j=1}^K \alpha_j$$

$$(11.12) \alpha_j \geq 0.$$

If we know every extreme point of X , then we can simply replace the “easy constraints” in (11.9) by (11.12) to get

$$\begin{aligned} \max \quad & \sum_{j=1}^K (\mathbf{c}^T \mathbf{v}_j) \alpha_j \\ \text{s.t.} \quad & \sum_{j=1}^K (A_H \mathbf{v}_j) \alpha_j = \mathbf{b}_H \\ & \sum_{j=1}^K \alpha_j = 1 \\ (11.13) \quad & \alpha_j \geq 0. \end{aligned}$$

The variables for (11.13) are the α_j 's, and once we've solved (11.13), we can get the optimal solution to (11.9) by using (11.12).

■ EXAMPLE 11.13

Consider the easy constraints for (11.10). By examining the feasible region given in [Figures 11.4](#) and [11.5](#) and adding slack variables, we can easily see that there are 12 total extreme points of X :

$$V = \left\{ (x_1, x_2, x_3, x_4, s_1, s_2, s_3) : \begin{array}{l} \mathbf{v}_1 = (0, 0, 0, 0, 6, 8, 6) \\ \mathbf{v}_2 = (0, 0, 0, 6, 6, 2, 0) \\ \mathbf{v}_3 = (0, 0, 2, 4, 6, 0, 0) \\ \mathbf{v}_4 = (0, 0, 4, 0, 6, 0, 2) \\ \mathbf{v}_5 = (0, 6, 0, 0, 0, 8, 6) \\ \mathbf{v}_6 = (0, 6, 0, 6, 0, 2, 0) \\ \mathbf{v}_7 = (0, 6, 2, 4, 0, 0, 0) \\ \mathbf{v}_8 = (0, 6, 4, 0, 0, 0, 2) \\ \mathbf{v}_9 = (2, 0, 0, 0, 0, 8, 6) \\ \mathbf{v}_{10} = (2, 0, 0, 6, 0, 2, 0) \\ \mathbf{v}_{11} = (2, 0, 2, 4, 0, 0, 0) \\ \mathbf{v}_{12} = (2, 0, 4, 0, 0, 0, 2) \end{array} \right\}.$$

Thus, we would need to compute $\mathbf{c}^T \mathbf{v}_j$ and $A_H \mathbf{v}_j$ for each extreme point, which can be done as

$$\hat{\mathbf{c}} = \begin{bmatrix} \mathbf{c}^T \mathbf{v}_1 \\ \mathbf{c}^T \mathbf{v}_2 \\ \vdots \\ \mathbf{c}^T \mathbf{v}_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 60 \\ 54 \\ 28 \\ 54 \\ 114 \\ 108 \\ 82 \\ 16 \\ 72 \\ 70 \\ 44 \end{bmatrix}$$

$$A_H [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \dots \mathbf{v}_{12}] = \begin{bmatrix} 0 & 18 & \dots & 20 \\ 0 & 6 & \dots & 6 \end{bmatrix}.$$

The linear program (11.10) would then be equivalent to

$$\max \quad 0\alpha_1 + 60\alpha_2 + \dots + 44\alpha_{12}$$

s.t.

$$\begin{bmatrix} 0 & 18 & \dots & 20 \\ 0 & 6 & \dots & 6 \\ 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{12} \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 1 \end{bmatrix}$$

$$(11.14) \quad \alpha_1, \alpha_2, \dots, \alpha_{12} \geq 0.$$

If $\boldsymbol{\alpha}$ is the optimal solution to (11.14), then

$$\mathbf{x} = \sum_{j=1}^{12} \alpha_j \mathbf{v}_j$$

is the optimal solution to (11.10).

It is probably obvious that we would not want to (and could not) write this linear program out completely—just finding all extreme points of X is a daunting task, and could result in millions (or more) of different variables. However, this seems to be a prime case where column generation could play a role.

Consider the linear program (11.13), which we will refer to as the *Master*

Linear Program. If our original linear program (11.9) had m hard constraints, the master linear program has $m + 1$ constraints; hence, we need to start with an initial basis matrix B_M with $m + 1$ rows and columns. As we did in Section 11.3, we first compute our dual variables

$$\mathbf{y}^T = [y_1 \ y_2 \ \dots \ y_m \ y_{m+1}] = \hat{\mathbf{c}}_B^T B^{-1},$$

where variables y_1, \dots, y_m correspond to the hard constraints and y_{m+1} corresponds to the constraint

$$\sum_{j=1}^K \alpha_j = 1.$$

We next need to check whether our current basic feasible solution is optimal, and if not, identify which column is to enter the basis. To do this, we consider our reduced costs

$$\begin{aligned}\bar{\mathbf{c}}_j &= \hat{\mathbf{c}}_j - \mathbf{y}^T \hat{\mathbf{a}}_j \\ &= \mathbf{c}^T \mathbf{v}_j - \mathbf{w}^T (A_H \mathbf{v}_j) - y_{m+1} \\ &= (\mathbf{c} - A_H^T \mathbf{w})^T \mathbf{v}_j - y_{m+1},\end{aligned}$$

where $\mathbf{w} = (y_1, \dots, y_m)$ is the vector of dual variables corresponding to the hard constraints. If every $\bar{\mathbf{c}}_j \leq 0$, we have an optimal solution to (11.13); otherwise, we can choose our entering variable from among those that have positive reduced cost. But if we don't know the columns $\hat{\mathbf{a}}_j$, how can we do this? Note that checking our reduced costs is equivalent to finding

$$z^* = \max_j \left\{ (\mathbf{c} - A_H^T \mathbf{w})^T \mathbf{v}_j - y_{m+1} \right\};$$

if $z^* \leq 0$, we are at the optimal solution. Since each \mathbf{v}_j is an extreme point of the feasible region of the easy constraints, and the simplex method outputs a basic feasible solution when it finds an optimal solution, we can obtain z^* by solving the linear program

$$\begin{aligned}z^* &= \max \quad (\mathbf{c} - A_H^T \mathbf{w})^T \mathbf{x} - y_{m+1} \\ \text{s.t.} \\ A_E \mathbf{x} &= \mathbf{b}_E \\ (11.15) \quad \mathbf{x} &\geq \mathbf{0}.\end{aligned}$$

Note that since y_{m+1} is a constant with regard to (11.15), it can be subtracted

from the optimal value at the end. If $z^* > 0$, we can use our optimal solution \mathbf{x}^* to generate the needed entering column of \hat{A} by

$$\hat{\mathbf{a}}_j = \begin{bmatrix} A_H \mathbf{x}^* \\ 1 \end{bmatrix}.$$

■ EXAMPLE 11.14

Consider the linear program (11.10), which we solve using Dantzig–Wolfe decomposition. Since there were two hard constraints, we need three extreme points for our basis; we will start with the basis formed by the extreme points

$$\mathbf{v}_9 = (2, 0, 0, 0, 0, 8, 6),$$

$$\mathbf{v}_5 = (0, 6, 0, 0, 0, 8, 6),$$

$$\mathbf{v}_7 = (0, 6, 2, 4, 0, 0, 0).$$

This generates the basis matrix

$$B = \begin{bmatrix} A_H \mathbf{v}_9 & A_H \mathbf{v}_5 & A_H \mathbf{v}_7 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 6 & 26 \\ 2 & 6 & 12 \\ 1 & 1 & 1 \end{bmatrix}$$

and objective coefficients

$$\hat{\mathbf{c}}_B = \begin{bmatrix} \mathbf{c}^T \mathbf{v}_9 \\ \mathbf{c}^T \mathbf{v}_5 \\ \mathbf{c}^T \mathbf{v}_7 \end{bmatrix} = \begin{bmatrix} 16 \\ 54 \\ 108 \end{bmatrix}.$$

We first compute our dual variables

$$\begin{aligned} \mathbf{y}^T &= \hat{\mathbf{c}}_B^T B^{-1} \\ &= \left[-\frac{3}{17} \quad \frac{163}{17} \quad -\frac{42}{17} \right], \end{aligned}$$

With $\mathbf{w} = (-\frac{3}{17}, \frac{163}{17})$. The objective coefficients for (11.15) are

$$\mathbf{c} - A_H^T \mathbf{w} = \begin{bmatrix} -\frac{21}{17} \\ -\frac{7}{17} \\ -\frac{32}{17} \\ \frac{16}{17} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

resulting in the linear program

$$z^* = \max \quad -\frac{21}{17}x_1 - \frac{7}{17}x_2 - \frac{32}{17}x_3 + \frac{16}{17}x_4 - \left(-\frac{42}{17}\right)$$

s.t.

$$3x_1 + x_2 + s_1 = 6$$

$$2x_3 + x_4 + s_2 = 8$$

$$x_3 + x_4 + s_3 = 6$$

$$x_1, x_2, x_3, x_4, s_1, s_2, s_3 \geq 0.$$

The optimal value is $z^* = \frac{138}{17}$ with solution $\mathbf{x}^* = \mathbf{v}_2 = (0, 0, 0, 6, 6, 2, 0)$. Thus, our entering column is

$$\begin{aligned} \hat{\mathbf{a}}_j &= \begin{bmatrix} A_H \mathbf{x}^* \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 18 \\ 6 \\ 1 \end{bmatrix}. \end{aligned}$$

Once we have identified a column \mathbf{a}_j to enter our basis matrix B , we next compute our direction vector \mathbf{d} by

$$\mathbf{d} = -B^{-1} \hat{\mathbf{a}}_j.$$

Given our current basic feasible solution $\alpha = B^{-1} \hat{\mathbf{b}}$, we can identify our leaving column by using the ratio test

$$\min \left\{ \frac{\alpha_j}{-d_j} : d_j < 0 \right\}.$$

At this point, we swap the entering column for the leaving column in the

basis matrix B and continue with the simplex method until we have reached the optimal solution.

■ EXAMPLE 11.15

Continuing on from Example 11.14, we first find our direction \mathbf{d} using

$$\begin{aligned}\mathbf{d} &= -B^{-1}\hat{\mathbf{a}}_j \\ &= -\begin{bmatrix} 0 & 6 & 26 \\ 0 & 6 & 12 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 18 \\ 6 \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{18}{17} \\ \frac{13}{17} \\ -\frac{12}{17} \end{bmatrix}.\end{aligned}$$

Since our current basic feasible solution is

$$\begin{aligned}\boldsymbol{\alpha} &= B^{-1}\hat{\mathbf{b}} \\ &= \begin{bmatrix} 0 & 6 & 26 \\ 0 & 6 & 12 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 20 \\ 10 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{17} \\ \frac{4}{17} \\ \frac{12}{17} \end{bmatrix},\end{aligned}$$

the ratio test gives

$$\min \left\{ \frac{\left(\frac{1}{17}\right)}{\left(\frac{18}{17}\right)}, \frac{\left(\frac{12}{17}\right)}{\left(\frac{12}{17}\right)} \right\} = \frac{1}{18},$$

indicating that the first column of B is to leave. This generates the new basis matrix

$$B = \begin{bmatrix} 18 & 6 & 26 \\ 6 & 6 & 12 \\ 1 & 1 & 1 \end{bmatrix}$$

and objective coefficients

$$\hat{\mathbf{c}}_B = \begin{bmatrix} 60 \\ 54 \\ 108 \end{bmatrix}.$$

In the second iteration, our vector \mathbf{y} is

$$\mathbf{y}^T = \hat{\mathbf{c}}_B^T B^{-1} = \begin{bmatrix} 1 & \frac{22}{3} & 7 \end{bmatrix},$$

giving $\mathbf{w} = (\frac{1}{2}, \frac{22}{3})$. This generates the objective coefficients

$$\mathbf{c} - A_H^T \mathbf{w} = \begin{bmatrix} -\frac{1}{3} \\ \frac{7}{6} \\ \frac{6}{6} \\ -\frac{7}{3} \\ \frac{7}{6} \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

resulting in the linear program

$$z^* = \max \quad -\frac{1}{3}x_1 + \frac{7}{6}x_2 - \frac{7}{3}x_3 + \frac{7}{6}x_4 - 7$$

s.t.

$$3x_1 + x_2 + s_1 = 6$$

$$2x_3 + x_4 + s_2 = 8$$

$$x_3 + x_4 + s_3 = 6$$

$$x_1, x_2, x_3, x_4, s_1, s_2, s_3 \geq 0.$$

The optimal solution is $\mathbf{x}^* = \mathbf{v}_6 = (0, 6, 0, 6, 0, 2, 0)$ with value $z^* = 7$, resulting in the entering column

$$\begin{aligned} \hat{A}_j &= \begin{bmatrix} A_H \mathbf{x}^* \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 24 \\ 12 \\ 1 \end{bmatrix}. \end{aligned}$$

We next compute the simplex direction

$$\mathbf{d} = -B^{-1} \hat{\mathbf{a}}_j$$

$$= - \begin{bmatrix} 18 & 6 & 26 \\ 6 & 6 & 12 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 24 \\ 12 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{6} \\ -\frac{1}{6} \\ -1 \end{bmatrix},$$

which, along with our current basic feasible solution

$$\alpha = B^{-1}\hat{b} = \begin{bmatrix} \frac{1}{18} \\ \frac{5}{18} \\ \frac{2}{3} \end{bmatrix},$$

gives the ratio test

$$\min \left\{ \frac{\left(\frac{5}{18}\right)}{\left(\frac{1}{6}\right)}, \frac{\left(\frac{2}{3}\right)}{1} \right\} = \frac{2}{3},$$

indicating that the third column exits the basis. Our new basis matrix is

$$B = \begin{bmatrix} 18 & 6 & 24 \\ 6 & 6 & 12 \\ 1 & 1 & 1 \end{bmatrix}$$

and objective coefficients

$$\hat{c}_B = \begin{bmatrix} 60 \\ 54 \\ 114 \end{bmatrix},$$

corresponding to the extreme points v_2, v_5, v_6 , respectively.

In the third iteration, our vector y is

$$y^T = \hat{c}_B^T B^{-1} = \begin{bmatrix} 1 & \frac{17}{2} & 0 \end{bmatrix},$$

and $w = (\frac{1}{2}, \frac{17}{2})$, which generates the objective coefficients

$$c - A_H^T w = \begin{bmatrix} -\frac{3}{2} \\ 0 \\ -\frac{7}{2} \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

This results in the linear program

$$\begin{aligned}
z^* = \max \quad & -\frac{3}{2}x_1 + 0x_2 - \frac{7}{2}x_3 + 0x_4 - 0 \\
\text{s.t.} \quad & \\
& 3x_1 + x_2 + s_1 = 6 \\
& 2x_3 + x_4 + s_2 = 8 \\
& x_3 + x_4 + s_3 = 6 \\
& x_1, x_2, x_3, x_4, s_1, s_2, s_3 \geq 0.
\end{aligned}$$

The optimal solution is $(0, 0, 0, 0, 6, 8, 6)$ with value $z^* = 0$. Thus, our current basic feasible solution

$$\alpha = B^{-1}\hat{\mathbf{b}} = \begin{bmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{2}{3} \end{bmatrix}$$

is optimal. Using these weights with the extreme points $\mathbf{v}_2, \mathbf{v}_5, \mathbf{v}_6$ gives the optimal solution

$$\mathbf{x}^* = \frac{1}{6}\mathbf{v}_2 + \frac{1}{6}\mathbf{v}_5 + \frac{2}{3}\mathbf{v}_6 = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 5 \\ 1 \\ 3 \\ 1 \end{bmatrix}$$

to our original linear program (11.10).

Dantzig–Wolfe decomposition is a useful approach to solving large linear programs when we have such “easy” and “hard” constraints. It is particularly useful in cases, such as our example problem (11.10), where the constraints are in a *Block Angular* design

$$\begin{aligned}
\max \quad & \mathbf{c}_1^T \mathbf{x}_1 + \mathbf{c}_2^T \mathbf{x}_2 + \cdots + \mathbf{c}_p^T \mathbf{x}_p \\
\text{s.t.} \quad & \\
& A_1^H \mathbf{x}_1 + A_2^H \mathbf{x}_2 + \cdots + A_p^H \mathbf{x}_p = \mathbf{b}^H \\
& A_1^E \mathbf{x}_1 = \mathbf{b}_1^E \\
& A_2^E \mathbf{x}_2 = \mathbf{b}_2^E \\
& \quad \ddots \quad \vdots \\
& A_p^E \mathbf{x}_p = \mathbf{b}_p^E \\
& \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p \geq 0.
\end{aligned}$$

In such problems, the reduced cost calculations in each iteration of the simplex method can be decomposed into p separate subproblems on only a subset of the original set of variables. While there are more problems to solve, typically each subproblem is easier to solve than the original one.

There are two aspects of this approach that we have not fully addressed here: finding an initial basic feasible solution to the master problem (11.13) and dealing with the case where the feasible region to the easy constraints is unbounded. In various cases, finding an initial basic feasible solution to (11.13) is straightforward (see Exercise 11.21), but may involve a Phase I-type approach. Similarly, the case with an unbounded feasible region can be handled with minor modifications to our above approach. For more information on each of these cases, the book by Bradley, Hax, and Magnanti [21] provides a good description.

11.5 PRIMAL–DUAL INTERIOR POINT METHOD

Until now all our algorithms for solving linear programs relied on moving from one basic feasible solution to another basic feasible solution along an edge of the feasible region. In the process, we always satisfied either primal or dual feasibility and complementary slackness, and only satisfied all three conditions at the end (assuming an optimal solution). There are approaches

that do neither of these; such algorithms travel through the interior of the feasible region (if one exists) and may not always satisfy two of the three conditions needed for optimality. In this section, we explore one such approach.

We assume that our primal linear program is in canonical form

$$\max \quad \mathbf{c}^T \mathbf{x}$$

s.t.

$$A\mathbf{x} = \mathbf{b}$$

$$(11.16) \quad \mathbf{x} \geq \mathbf{0}$$

and its dual is given by

$$\min \quad \mathbf{b}^T \mathbf{y}$$

s.t.

$$A^T \mathbf{y} - \mathbf{w} = \mathbf{c}$$

$$(11.17) \quad \mathbf{w} \geq \mathbf{0},$$

where \mathbf{w} is the dual surplus variables. For this problem, the complementary slackness condition ensures that $x_k w_k = 0$ for all $k \in \{1, \dots, n\}$. We have an optimal primal solution \mathbf{x}^* and an optimal dual solution $(\mathbf{y}^*, \mathbf{w}^*)$ if and only if these solutions satisfy

$$A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}$$

$$A^T \mathbf{y} - \mathbf{w} = \mathbf{c}, \quad \mathbf{w} \geq \mathbf{0}$$

$$x_k w_k = 0, \quad k \in \{1, \dots, n\}.$$

Interior point methods travel through the interior of the feasible region toward an optimal solution. They maintain a *strictly feasible* solution throughout, in which a feasible primal solution \mathbf{x} satisfies $x_k > 0$, and the algorithm converges to an optimal solution; hence, if there is a unique solution to a linear program (which would then occur at a basic feasible solution), interior point methods obtain this solution only in a limit. One way to formulate this is to assume we have a strictly feasible solution $\mathbf{x} > \mathbf{0}$ to the primal and feasible solution (\mathbf{y}, \mathbf{w}) to the dual where $\mathbf{w} > \mathbf{0}$, so that these solutions satisfy

$$A\mathbf{x} = \mathbf{b}$$

$$A^T \mathbf{y} - \mathbf{w} = \mathbf{c}.$$

However, since the solutions are strictly feasible, complementary slackness conditions do not hold; thus, we restrict our solutions \mathbf{x} and \mathbf{w} to satisfy

$$(11.18) \quad x_k w_k = \mu > 0, \quad k = 1, \dots, n,$$

so that our solutions $\mathbf{x}(\mu)$ and $\mathbf{w}(\mu)$ (and $\mathbf{y}(\mu)$) are really functions of μ ; note that (11.18) implies that the complementary slackness condition is now $\mathbf{x}^T \mathbf{w} = n\mu$. We can write this approximate complementary slackness condition in a matrix form. If we let X be the diagonal matrix where $X_{kk} = x_k$ and $X_{ij} = 0$ if $i \neq j$ and W be the diagonal matrix where $W_{kk} = w_k$ and $W_{ij} = 0$ if $i \neq j$, then we can rewrite (11.18) as

$$XW\mathbf{e} = \mu\mathbf{e},$$

where \mathbf{e} is a vector of all 1's, so that the solution vector $(\mathbf{x}(\mu), \mathbf{y}(\mu), \mathbf{w}(\mu))$ satisfies

$$(11.19a) \quad A\mathbf{x}(\mu) = \mathbf{b}$$

$$(11.19b) \quad A^T \mathbf{y}(\mu) - \mathbf{w}(\mu) = \mathbf{c}$$

$$(11.19c) \quad XW\mathbf{e} = \mu\mathbf{e}.$$

As $\mu \rightarrow 0$, any limiting solution to (11.19) gives an optimal solution to our primal–dual pair.

Of course, finding feasible solutions to both the primal and the dual problems that also satisfy (11.19c) seems difficult, so we shall approximate a solution to this equation. Suppose that, for $\mu > 0$, we have a solution to (11.19a) and (11.19b) where (11.19c) is not necessarily satisfied but $\mathbf{x}(\mu) > \mathbf{0}$ and $\mathbf{w}(\mu) > \mathbf{0}$. We would like to find directions \mathbf{d}^x , \mathbf{d}^y , and \mathbf{d}^w so that $\mathbf{x} + \mathbf{d}^x$ satisfies (11.19a), $(\mathbf{y} + \mathbf{d}^y, \mathbf{w} + \mathbf{d}^w)$ satisfies (11.19b), and $(\mathbf{x} + \mathbf{d}^x, \mathbf{w} + \mathbf{d}^w)$ approximately satisfies (11.19c). In order for $\mathbf{x} + \mathbf{d}^x$ to satisfy (11.19a) and $(\mathbf{y} + \mathbf{d}^y, \mathbf{w} + \mathbf{d}^w)$ to satisfy (11.19b), we require that

$$A\mathbf{d}^x = \mathbf{0}$$

$$A^T \mathbf{d}^y - \mathbf{d}^w = \mathbf{0}.$$

We had previously seen this calculation in Chapter 6 when we determined feasible directions. Now consider the equation

$$(x_k + d_k^x)(w_k + d_k^w) = \mu$$

from (11.19c). Expanding this gives

$$x_k w_k + x_k d_k^w + w_k d_k^x + d_k^x d_k^w = \mu,$$

which is a nonlinear equation due to the term $d_k^x d_k^w$. We can generate an approximate solution by removing this term and solving

$$x_k d_k^w + w_k d_k^x = \mu - x_k w_k$$

for d_k^x and d_k^w . This system of equations can be written in matrix form as

$$X\mathbf{d}^w + W\mathbf{d}^x = \mu\mathbf{e} - XW\mathbf{e}.$$

Thus, we can produce a new solution($\mathbf{x} + \mathbf{d}^x$, $\mathbf{y} + \mathbf{d}^y$, $\mathbf{w} + \mathbf{d}^w$) that solves (11.19a) and (11.19b) while better approximating (11.19c) for our fixed μ by solving the system of equations

$$(11.20a) A\mathbf{d}^x = \mathbf{0}$$

$$(11.20b) A^T\mathbf{d}^y - \mathbf{d}^w = \mathbf{0}$$

$$(11.20c) X\mathbf{d}^w + W\mathbf{d}^x = \mu\mathbf{e} - XW\mathbf{e}.$$

This yields

$$(11.21a) \mathbf{d}^y = \left(AW^{-1}XA^T \right)^{-1} AW^{-1}\mathbf{v}(\mu)$$

$$(11.21b) \mathbf{d}^w = A^T\mathbf{d}^y$$

$$(11.21c) \mathbf{d}^x = W^{-1}(\mathbf{v}(\mu) - X\mathbf{d}^w)$$

where $\mathbf{v}(\mu) = \mu\mathbf{e} - XW\mathbf{e}$ (see Exercise 11.23).

■ EXAMPLE 11.16

Consider the linear program we examined in Chapter 8

$$\max \quad 13x_1 + 5x_2$$

s.t.

$$4x_1 + x_2 + s_1 = 24$$

$$x_1 + 3x_2 + s_2 = 24$$

$$3x_1 + 2x_2 + s_3 = 23$$

$$(11.22) \quad x_1, x_2, s_1, s_2, s_3 \geq 0.$$

Its dual is

$$\begin{aligned}
\min \quad & 24y_1 + 24y_2 + 23y_3 \\
\text{s.t.} \quad & \\
& 4y_1 + y_2 + 3y_3 - w_1 = 13 \\
& y_1 + 3y_2 + 2y_3 - w_2 = 5 \\
& y_1 - w_{s_1} = 0 \\
& y_2 - w_{s_2} = 0 \\
& y_3 - w_{s_3} = 0
\end{aligned}$$

$$(11.23) \quad w_1, w_2, w_{s_1}, w_{s_2}, w_{s_3} \geq 0.$$

Suppose, we have the strictly feasible solutions $\mathbf{x}^0 = (2, 2, 14, 16, 13)$ and $(\mathbf{y}^0, \mathbf{w}^0) = (3, 2, 1, 4, 6, 3, 2, 1)$. If we let

$$X = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 14 & 0 & 0 \\ 0 & 0 & 0 & 16 & 0 \\ 0 & 0 & 0 & 0 & 13 \end{bmatrix}, \quad W = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

we get

$$XW\mathbf{e} = \begin{bmatrix} 8 \\ 12 \\ 42 \\ 32 \\ 13 \end{bmatrix}.$$

Suppose, we let $\mu = 20$. Solving (11.20) using (11.21) gives (rounded to six digits)

$$\mathbf{d}^x = \begin{bmatrix} 1.592259 \\ 0.871045 \\ -7.240081 \\ -4.205393 \\ -6.518867 \end{bmatrix}, \quad \mathbf{d}^y = \begin{bmatrix} -0.019983 \\ -0.224326 \\ 1.039913 \end{bmatrix}, \quad \mathbf{d}^w = \begin{bmatrix} 2.815482 \\ 1.386866 \\ -0.019983 \\ -0.224326 \\ 1.039913 \end{bmatrix}.$$

Our new solutions would be $\mathbf{x}^0 + \mathbf{d}^x = (3.592259, 2.871045, 6.759919, 11.794607, 6.481133)$, $\mathbf{y}^0 + \mathbf{d}^y = (2.980017, 1.775674, 2.039913)$, $\mathbf{w}^0 + \mathbf{d}^w = (6.815482, 7.386866, 2.980017, 1.775674, 2.039913)$. If we check $XW\mathbf{e}$ with

these updated solutions, we get

$$XWe = \begin{bmatrix} 24.482998 \\ 21.208022 \\ 20.144676 \\ 20.943378 \\ 13.220947 \end{bmatrix}$$

which is much closer to $\mu\mathbf{e} = 20\mathbf{e}$ than our initial estimate.

We could continue with this approach to find better estimates to $\mu\mathbf{e}$, but our real goal is to solve the linear program by decreasing μ to 0. It seems reasonable that our next step reduces μ to

$$\mu^{new} \leftarrow \theta\mu^{old},$$

where $0 < \theta < 1$. If θ is appropriately chosen, then by iteratively decreasing μ by a factor of θ and finding new solutions \mathbf{x} , \mathbf{y} , \mathbf{w} , we will not only maintain $\mathbf{x} > \mathbf{0}$ and $\mathbf{w} > \mathbf{0}$ at each iteration, but we will also converge to an optimal solution to (11.22) and its dual (11.23). Unfortunately, such values of θ tend to be close to 1 so that the convergence is very slow.

What we would like to do is decrease μ more rapidly, but then we have to worry about \mathbf{x} and \mathbf{w} remaining strictly feasible.

■ EXAMPLE 11.17

Suppose, in Example 11.16 we decrease μ by a factor of 10 from 20 to 2 and start from our current estimates $\mathbf{x}^1 = (3.592259, 2.871045, 6.759919, 11.794607, 6.481133)$, $\mathbf{y}^1 = (2.980017, 1.775674, 2.039913)$, $\mathbf{w}^1 = (6.815482, 7.386866, 2.980017, 1.775674, 2.039913)$. If we again use (11.21) to compute our directions, we get the directions

$$\mathbf{d}^x = \begin{bmatrix} 0.912667 \\ 0.094136 \\ -3.74480 \\ -1.195074 \\ -2.926272 \end{bmatrix}, \quad \mathbf{d}^y = \begin{bmatrix} -1.033311 \\ -1.426187 \\ -0.810291 \end{bmatrix}, \quad \mathbf{d}^w = \begin{bmatrix} -7.990304 \\ -6.932455 \\ -1.033311 \\ -1.426187 \\ -0.810291 \end{bmatrix},$$

which generate the new solutions $\mathbf{x}^2 = \mathbf{x}^1 + \mathbf{d}^x = (4.504926, 2.965180, 30.15116, 10.599533, 3.554861)$, $\mathbf{y}^2 = \mathbf{y}^1 + \mathbf{d}^y = (1.946707, 0.349487, 1.226214)$, $\mathbf{w}^2 = \mathbf{w}^1 + \mathbf{d}^w = (-1.174822, 0.454410, 1.946707, 0.349487)$,

1.226214). Note that $w_1^2 < 0$, violating our bounds.

To fix this, we adjust our step size for each solution to ensure that we remain strictly feasible by calculating

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} + \alpha_p \mathbf{d}^x,$$

$$\mathbf{y}^{\text{new}} = \mathbf{y}^{\text{old}} + \alpha_d \mathbf{d}^y,$$

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \alpha_d \mathbf{d}^w,$$

where α_p and α_d are chosen using a modified ratio test

$$\alpha_p = \beta \min \left\{ \frac{x_k^{\text{old}}}{-d_k^x} : d_k^x < 0 \right\}$$

$$\alpha_d = \beta \min \left\{ \frac{w_k^{\text{old}}}{-d_k^w} : d_k^w < 0 \right\},$$

for β close to 1 because we want to remain strictly feasible. If α_p or α_d becomes bigger than 1, we just reset them to 1.

■ EXAMPLE 11.18

If in Example 11.17 we use the calculated directions \mathbf{d}^x , \mathbf{d}^y , and \mathbf{d}^w and we let $\beta = 0.9995$, we would find that

$$\alpha_p = 0.9995 \frac{x_3^1}{d_3^x} = 1.804244,$$

$$\alpha_d = 0.9995 \frac{w_1^1}{d_1^w} = 0.852543.$$

We then reset $\alpha_p = 1$, and so our new estimates would be $\mathbf{x}^2 = \mathbf{x}^1 + \alpha_p \mathbf{d}^x = (4.504926, 2.965180, 3.015116, 10.599533, 3.554861)$, $\mathbf{y}^2 = \mathbf{y}^1 + \alpha_d \mathbf{d}^y = (2.099076, 0.559789, 1.349105)$, and $\mathbf{w}^2 = \mathbf{w}^1 + \alpha_d \mathbf{d}^w = (0.003408, 1.476653, 2.099076, 0.559789, 1.349105)$.

We then continue to decrease μ until we have done a predetermined number of iterations or the components of $\mathbf{x}^T \mathbf{w}$ are close to 0. This approach to solving linear programs is known as the **Primal–Dual Interior Point Method**, and is given in Algorithm 11.4. It was one of the first successfully implemented interior point algorithms, and is the basis for many current

implementations.

Algorithm 11.4 Primal-Dual Interior Point Algorithm

Step 0: Initialization. Start with a primal linear program in canonical form. Identify a strictly feasible solution $\mathbf{x}^{(0)}$ to the primal problem and a strictly feasible solution $(\mathbf{y}^0, \mathbf{w}^0)$ to the dual problem (after adding surplus variables \mathbf{w}). Fix parameter $0 < \beta < 1$. Set index $t \leftarrow 0$.

Step 1: Obtain Directions. Let X be the diagonal matrix generated from \mathbf{x}^k and W the diagonal matrix generated from \mathbf{w}^k . Compute the appropriate directions $\mathbf{d}^y, \mathbf{d}^w, \mathbf{d}^x$ using the calculations from (11.21):

$$\begin{aligned}\mathbf{d}^y &= (AW^{-1}XA^T)^{-1} AW^{-1}\mathbf{v}(\mu) \\ \mathbf{d}^w &= A^T\mathbf{d}^y \\ \mathbf{d}^x &= W^{-1}(\mathbf{v}(\mu) - X\mathbf{d}^w),\end{aligned}$$

where $\mathbf{v}(\mu) = \mu\mathbf{e} - X\mathbf{w}\mathbf{e}$.

Step 2: Compute Step Sizes. Compute the step sizes

$$\begin{aligned}\alpha_p &= \min \left\{ 1, \beta \min \left\{ \frac{x_k^{\text{old}}}{-d_k^x} : d_k^x < 0 \right\} \right\} \\ \alpha_d &= \min \left\{ 1, \beta \min \left\{ \frac{w_k^{\text{old}}}{-d_k^w} : d_k^w < 0 \right\} \right\}. \quad (11.24)\end{aligned}$$

Step 3: Update Solutions. Compute new solutions

$$\begin{aligned}\mathbf{x}^{k+1} &= \mathbf{x}^k + \alpha_p \mathbf{d}^x \\ \mathbf{y}^{k+1} &= \mathbf{y}^k + \alpha_d \mathbf{d}^y \\ \mathbf{w}^{k+1} &= \mathbf{w}^k + \alpha_d \mathbf{d}^w.\end{aligned}$$

Step 4: Update Parameters and Check Stopping Conditions. Update μ and check stopping conditions. If μ “small enough,” stop; otherwise, increase k by 1 and return to Step 1.

■ EXAMPLE 11.19

If we continue from Example 11.18, decreasing μ at each iteration by a factor of 10, we get the following complementary slackness calculations:

μ	$\mathbf{x}^T \mathbf{w}$
2×10^1	100
2×10^0	21.452232
2×10^{-1}	2.452336
2×10^{-2}	0.314051

2×10^{-3}	0.01
2×10^{-4}	0.001
2×10^{-5}	0.0001
2×10^{-6}	0.00001

Since complementary slackness is approaching 0, our solutions are approaching the optimal solutions to both the primal and the dual problems. After eight iterations, our solutions are

$$\mathbf{x} = \begin{bmatrix} 5.000000 \\ 3.999999 \\ 0.000000 \\ 7.000002 \\ 0.000001 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 2.200000 \\ 0.000000 \\ 1.400000 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 0.000000 \\ 0.000000 \\ 2.200000 \\ 0.000000 \\ 1.400000 \end{bmatrix}.$$

Recall that the optimal basic feasible solution (found in Chapter 8) is $\mathbf{x} = (5, 4, 0, 7, 0)$, with the corresponding optimal dual solution $\mathbf{y} = (2.2, 0, 1.4)$.

For large-scale linear programs, interior point methods can take less time to solve problems than the simplex method. However, these approaches do have their drawbacks. First, each iteration is computationally intensive, requiring the solution of multiple systems of equations (for the inverse calculations), and round-off errors can become an issue. Second, we started with a strictly feasible solution to both the primal and the dual problems, but these are not easy to identify. Many optimization packages incorporate an approach where an initial infeasible solution to both problems is allowed. Third, because $\mu > 0$, we are not typically at a basic feasible solution. This, however, is not a problem since there are methods to generate a basic feasible solution from the optimal solution of an interior point method.

Summary

In this chapter we derived and examined new algorithms for solving special cases of linear programs. In each case, we exploited both the simplex method and linear programming duality to enhance our abilities to solve problems. It hopefully has become apparent that duality enables us to broaden our solution techniques for linear programs. In fact, this is one of the strengths of duality—giving us fresh perspectives on the simplex method and providing tools to improve the basic algorithm to specialized cases.

EXERCISES

11.1 Consider the linear program

$$\max \quad 4x + 3y$$

s.t.

$$2x + y \leq 10$$

$$x + y \leq 8$$

$$x \leq 4$$

$$x, y \geq 0,$$

whose optimal solution is $(x, y) = (2, 6)$. If we add the constraint $3x + 2y \leq 17$, resolve it using the dual simplex method.

11.2 Consider the linear program

$$\max \quad 5x - y$$

s.t.

$$-x + 2y \leq 30$$

$$x + y \leq 27$$

$$4x - y \leq 43$$

$$x - y \leq 7$$

$$x, y \geq 0,$$

whose optimal solution is $(x, y) = (14, 13)$. If we add the constraint $13x - y \leq 127$, resolve it using the dual simplex method:

11.3 Solve the following problem using the dual simplex method.

$$\max \quad -8x_1 - 3x_2 - 6x_3 - 5x_4$$

s.t.

$$2x_1 - 3x_2 + 5x_3 + 4x_4 \geq 21$$

$$7x_1 + 2x_2 + 6x_3 - 2x_4 \leq 30$$

$$4x_1 + 5x_2 - 3x_3 + 2x_4 \geq 17$$

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

11.4 Solve the following problem using the dual simplex method:

$$\begin{aligned}
\min \quad & 4x_1 + 7x_2 + 5x_3 \\
\text{s.t.} \quad & 2x_1 + x_2 + x_3 \geq 5 \\
& 3x_1 + 2x_2 + 4x_3 \geq 7 \\
& x_1, x_2, x_3 \geq 0.
\end{aligned}$$

11.5 Finish the transportation problem begun in Example 11.9.

11.6 Solve the following transportation problem using the transportation simplex method:

$$\begin{aligned}
\min \quad & 8x_{11} + 6x_{12} + 12x_{13} + 9x_{14} + 9x_{21} + 12x_{22} + 13x_{23} + 7x_{24} \\
& + 14x_{31} + 9x_{32} + 16x_{33} + 5x_{34}
\end{aligned}$$

s.t.

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 35$$

$$x_{21} + x_{22} + x_{23} + x_{24} \leq 50$$

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 40$$

$$x_{11} + x_{21} + x_{31} \geq 40$$

$$x_{12} + x_{22} + x_{32} \geq 20$$

$$x_{13} + x_{23} + x_{33} \geq 30$$

$$x_{14} + x_{24} + x_{34} \geq 30$$

$$x_{ij} \geq 0, \text{ for all } i, j.$$

11.7 Prove Lemma 11.1: If A is the constraint matrix of the transportation problem (11.5), then $\text{Rank}(A) = m + n - 1$. Furthermore, an $(m + n - 1, m + n - 1)$ submatrix B has full row rank if and only if the submatrix B corresponds to the node–arc matrix of a spanning tree of the network.

11.8 Given the transportation tableau

20	45	35	10		30
35	40	50	20		30
30	20	15	25		20
30	25	20	5		

use the minimum cost method to obtain an initial solution that is

degenerate. Show that the cells x_{12} and x_{24} cannot become basic variables, while any of the other empty cells may become basic.

11.9 (This problem was first given in Exercise 2.32.) Indiana Power is a local cooperative that supplies the power needs of four cities using three power plants. The potential supply from each power plant and the peak power demand for each city (each given in millions of kilowatt/hours of electricity) are given below.

Supply		Demand	
		City 1	115
Plant 1	85	City 2	70
Plant 2	115	City 3	65
Plant 3	100	City 4	50

Finally, the cost (in dollars) of sending a million kwh from each plant to each city is given below.

	City 1	City 2	City 3	City 4
Plant 1	10	7	10	6
Plant 2	7	12	16	9
Plant 3	12	8	13	7

Indiana Power wants to find the lowest cost method for meeting the demand of the four cities. Solve the transportation problem to minimize the cost to Indiana Power.

11.10 Solve the following transportation problem:

Supply locations	Demand Locations			Capacities
	1	2	3	
1	30	25	35	50
2	25	35	20	30
3	25	20	30	20
4	20	15	25	30
Demand	50	40	40	

11.11 Consider the cutting stock problem with a raw width of 50 inches and order summary

74 finals of width 10 inches

96 finals of width 15 inches

136 finals of width 18 inches

100 finals of width 22 inches

120 finals of width 27 inches.

(a) How many possible patterns are there?

(b) Solve this cutting stock problem using column generation. Begin with an initial basic feasible solution where the columns of B are the patterns consisting of only one width and that pattern has the maximum number of each width (e.g., there is a pattern using exactly three of width 15 inches). How many patterns did you create before finding the optimal set of patterns?

11.12 Consider the cutting stock problem with raw width 100 inches and order summary

103 finals of width 42 inches

210 finals of width 34 inches

195 finals of width 28 inches

251 finals of width 13 inches.

Solve this cutting stock problem using column generation. The answer will not be integral. What can be done to generate an integer solution? Would this approach generate the optimal integer solution? Explain.

11.13 For the cutting stock problem, we solve knapsack problem in each iteration, stopping only when the optimal value $z^* \leq 1$. Is it possible for $z^* < 1$? Explain.

11.14 Solve the following linear program using Dantzig–Wolfe Decomposition:

$$\begin{aligned}
\max \quad & 3x_1 + 5x_2 + 7x_3 + 4x_4 \\
\text{s.t.} \quad & \\
& 2x_1 + x_2 + 3x_3 + 2x_4 = 15 \\
& x_1 + x_2 + x_3 + x_4 = 10 \\
& x_1 + x_2 \leq 7 \\
& 4x_1 + 3x_2 \leq 24 \\
& 2x_3 + x_4 \leq 10 \\
& x_3 + x_4 \leq 8 \\
& x_3 \leq 4 \\
& x_1, x_2, x_3, x_4 \geq 0.
\end{aligned}$$

Use the first two constraints as the “hard” constraints.

11.15 Solve the following linear program using Dantzig–Wolfe Decomposition.

$$\begin{aligned}
\max \quad & 3x_1 + 5x_2 + 7x_3 + 4x_4 \\
\text{s.t.} \quad & \\
& 2x_1 + x_2 + 3x_3 + 2x_4 = 15 \\
& x_1 + x_2 + x_3 + x_4 = 10 \\
& x_1 + x_2 \leq 7 \\
& 4x_1 + 3x_2 \leq 24 \\
& 2x_3 + x_4 \leq 10 \\
& x_3 + x_4 \leq 8
\end{aligned}$$

Use the first two constraints as the “hard” constraints.

11.16 Solve the problem

$$\begin{aligned}
\max \quad & 3x + 2y \\
\text{s.t.} \quad & \\
& 2x + y \leq 10 \\
& x + y \leq 8 \\
& x \leq 4 \\
& x, y \geq 0
\end{aligned}$$

using the primal–dual interior point algorithm. Use (1, 1) as the initial

primal solution and $(2, 2, 1)$ as the initial dual solution. Begin with $\mu = 10$ and $\theta = 0.1$, and do eight iterations.

11.17 Solve the problem

$$\min \quad 4x_1 + 7x_2 + 5x_3$$

s.t.

$$2x_1 + x_2 + x_3 \geq 5$$

$$3x_1 + 2x_2 + 4x_3 \geq 7$$

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

using the primal–dual interior point algorithm. Use $(2, 2, 1)$ as the initial primal solution and $(0.5, 0.5)$ as the initial dual solution. Begin with $\mu = 10$ and $\theta = 0.1$, and do eight iterations. (*Hint:* can we treat the dual problem as the “primal”?)

11.18 Given a linear program in canonical form

$$\max \quad \mathbf{c}^T \mathbf{x}$$

s.t

$$A\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} \geq \mathbf{0},$$

and its dual problem

$$\min \quad \mathbf{b}^T \mathbf{y}$$

s.t

$$A^T \mathbf{y} \geq \mathbf{c},$$

suppose we have a primal basic solution $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)$ that generates a dual feasible solution $\mathbf{y} = \mathbf{c}_B^T B^{-1}$. Show that if we solve the dual problem using the simplex method, as we did in Section 11.1, we have

1. The reduced cost vector $\hat{\bar{c}}_B$ associated with the dual nonbasic variables \mathbf{w}_B is

$$\hat{\bar{c}}_B = \mathbf{x}_B,$$

which are the values of the primal basic variables.

2. The components of the simplex direction \mathbf{d} corresponding to the basic dual variables \mathbf{w}_N is

$$\mathbf{d}_{wN}^T = \mathbf{e}_k^T B^{-1} N,$$

which is the k th row of the matrix $B^{-1}N$.

3. The ratio test becomes the calculation

$$\lambda_{\max} = \min \left\{ \frac{\bar{c}_j}{d_{wj}} : d_{wj} < 0 \right\},$$

where \bar{c}_j is the reduced cost of the primal nonbasic variable x_j and d_{wj} is the component of the simplex direction vector \mathbf{d}_{wN} corresponding to the dual basic variable w_j .

11.19 In the dual simplex method, we need to construct the direction vector

$$\mathbf{d}^T = \mathbf{e}_k^T B^{-1} N$$

in Step 2. How can we determine \mathbf{d}^T without directly calculating B^{-1} ?

11.20 Why is it true that in the dual simplex method, when choosing an entering variable, if all \bar{a}_{kj} are nonpositive, the problem is infeasible?

11.21 Suppose we are given the linear program

$$\max \quad \mathbf{c}^T \mathbf{x}$$

s.t.

$$A_H \mathbf{x} \leq \mathbf{b}_H$$

$$A_E \mathbf{x} \leq \mathbf{b}_E$$

$$\mathbf{x} \geq \mathbf{0},$$

where $\mathbf{b}_H \geq \mathbf{0}$ and $\mathbf{b}_E \geq \mathbf{0}$ and we wish to solve this problem using Dantzig–Wolfe decomposition. If V is the matrix whose columns are the extreme points of $\mathbf{x} \in \mathbb{R}^n : A_E \mathbf{x} \leq \mathbf{b}_E, \mathbf{x} \geq \mathbf{0}$, the master problem is then

$$\max \quad \mathbf{c}_M^T \boldsymbol{\alpha}$$

s.t.

$$(A_H V) \boldsymbol{\alpha} + s = \mathbf{b}_H$$

$$\mathbf{e}^T \boldsymbol{\alpha} = 1$$

$$\boldsymbol{\alpha}, s \geq \mathbf{0}.$$

Show that an initial basis matrix B to this master problem is $B = I$.

11.22 Show that if A is an $m \times n$ matrix with rank m , then for any diagonal $n \times n$ matrix D whose diagonal entries are all nonzero (so that D^{-1} exists) the matrix $(ADA^T)^{-1}$ exists.

11.23 Suppose A is an $m \times n$ matrix with rank m and X and W are both diagonal $n \times n$ matrices with nonzero diagonal elements, implying that X^{-1} and W^{-1} both exist. Use Exercise 11.22 to derive (not verify!) the directions

$$\begin{aligned}\Delta \mathbf{y} &= \left(AW^{-1} X A^T \right)^{-1} AW^{-1} \mathbf{v}(\mu) \\ \Delta \mathbf{w} &= A^T \Delta \mathbf{y} \\ \Delta \mathbf{x} &= W^{-1} (\mathbf{v}(\mu) - X \Delta \mathbf{w}),\end{aligned}$$

where $\mathbf{v}(\mu) = \mu \mathbf{e} - X W \mathbf{e}$, solve the system of equations

$$\begin{aligned}A \Delta \mathbf{x} &= \mathbf{0} \\ A^T \Delta \mathbf{y} - \Delta \mathbf{w} &= \mathbf{0} \\ X \Delta \mathbf{w} + W \Delta \mathbf{x} &= \mu \mathbf{e} - X W \mathbf{e}.\end{aligned}$$

CHAPTER 12

NETWORK OPTIMIZATION ALGORITHMS

12.1 INTRODUCTION TO NETWORK OPTIMIZATION

In previous chapters, we concentrated on solving linear programs through a variety of approaches. Each approach was based upon ideas from continuous optimization. In fact, we have even used these ideas to design an algorithm (transportation simplex) for a network optimization problem. For these algorithms, we have first derived optimality conditions and then designed approaches to satisfy them.

As we saw in Chapter 5, when we first encountered the minimum spanning tree problem, designing algorithms based upon optimality conditions is natural for some discrete problems. In this chapter, we explore such approaches for three classic network optimization problems: the shortest path problem, the maximum flow problem, and the minimum cost network flow problem. For each of these problems, we take a graph-theoretic approach to derive an optimality condition, using a sample problem to illustrate the ideas. We then design algorithms based upon these optimality conditions. We will also explore how we can redesign the bounded simplex method for the minimum cost network flow problem, similar to our work on the transportation problem in Section 11.2.

12.2 SHORTEST PATH

PROBLEMS

The shortest path problem is one of the fundamental combinatorial optimization problems, primarily due to the fact that it is often a subproblem in algorithms for various optimization problems. They also have many important applications; we have already seen in Chapter 2 how shortest path problems occur in equipment replacement. In this section, we explore the shortest path problem first by deriving an optimality condition for it and then by using this condition to produce various algorithms. Throughout this section, we examine issues related to algorithm design and its analysis.

To begin, it's useful to formally state the problem.

Shortest Path Problem Given a directed network $G = (V, A)$ with lengths c_{ij} associated with each arc $(i, j) \in A$ and a root node s , the *shortest path problem* seeks to determine for all nonsource nodes $i \in V - \{s\}$ a minimum length directed path from s to i and the length $d(i)$ of this path, where the length of a directed path P is the sum of the lengths of all arcs (j, k) on the path P , and there are no repeated vertices in P .

Recall from Section 2.9 that the shortest path problem can be formulated as a special case of the minimum cost network flow problem by sending $n - 1$ units of flow from s and with every other node $i \in V - \{s\}$ receiving one unit of flow. We can then state this problem as the linear program

$$\begin{aligned}
 & \min \quad \sum_{(i,j) \in A} c_{ij} x_{ij} \\
 & \text{s.t.} \\
 & \quad \sum_{j:(i,j) \in A} x_{ij} - \sum_{k:(k,i) \in A} x_{ki} = \begin{cases} n-1, & i = s, \\ -1, & i \in V - \{s\}, \end{cases} \\
 & \quad x_{ij} \geq 0, \quad (i, j) \in A.
 \end{aligned} \tag{12.1}$$

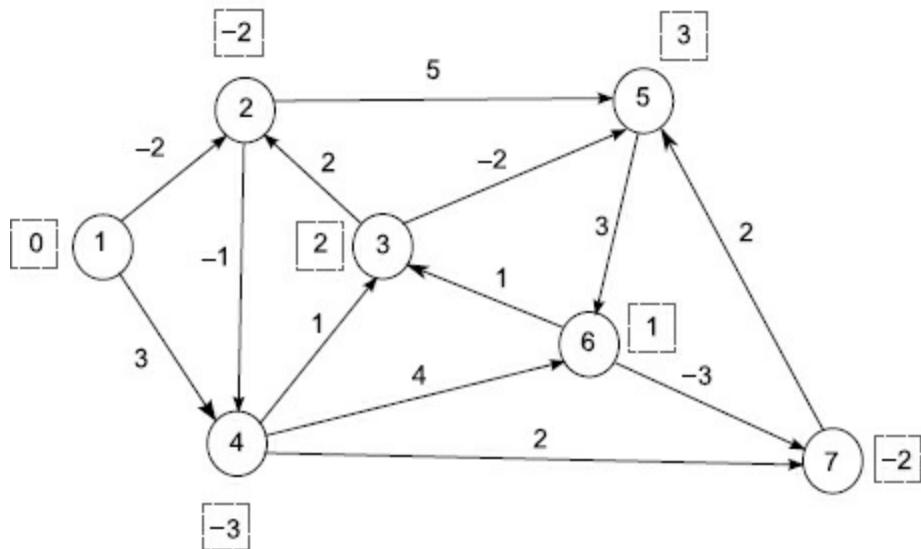
As with most problems, we make several assumptions in order to simplify our work. Note that some of these assumptions are not very restrictive.

Assumptions

1. For each $(i, j) \in A$, c_{ij} is an integer. We will allow each c_{ij} to be positive, negative, or zero.
2. The network G contains a directed path from s to every node $i \in V$. (If needed, we can add an arc (s, i) with extremely large length c_{si} if no such path initially existed.)
3. The network G does not contain a negative cycle (i.e., a directed cycle of negative length).

In order to determine necessary and sufficient conditions for a problem, it is often useful to examine a specific example and use this to help derive the conditions. With this in mind, consider the instance given in [Figure 12.1](#), where the boxes near each node j indicates the length $d(j)$ of some path from node $s = 1$ to node j . Is this set of lengths $d(j)$ optimal? For this problem, the answer is no because we can improve the length $d(3)$ from 2 to -2 by considering the path from node 1 to node 4 (through node 2) of length -3 and then adding the arc $(3, 4)$ of length 1. Note that a similar change could be made for the length of node 5.

[FIGURE 12.1](#) Shortest path example.



How can we describe such a (necessary) condition? For node 3, we noted that

$$d(3) > d(4) + c_{43}.$$

Similarly, we can see that $d(5) > d(7) + c_{75}$. From this analysis, we get the following necessary condition.

Lemma 12.1 *Given the set of lengths $d(j)$ associated with the length of a directed path from node s to node j for each $j \in S$ (we assume that $d(s) = 0$), if these lengths are the lengths of some shortest path from node s to j , then*

$$(12.2) \quad d(5) > d(7) + c_{75}.$$

for all $(i, j) \in A$.

Proof Suppose condition (12.2) is not satisfied for some nodes i, j . This implies that $d(j) > d(i) + c_{ij}$. For each node k , let P_k denote a directed path from node s to node k with length $d(k)$. Consider the path $P = P_i \cup \{(i, j)\}$. This path has length $d(i) + c_{ij}$, which is smaller than $d(j)$. Hence, $d(j)$ cannot be the length of some shortest path to j from s .

Is condition (12.2) a sufficient condition for a shortest path? The answer is yes.

Shortest Path Optimality Condition Let $d(j)$ denote the length of some directed path from source node s to j for each $j \in N$ (we assume that $d(s) = 0$). Then $d(j)$ denotes the lengths on the shortest path from s to j if and only if

$$(12.3) \quad d(j) \leq d(i) + c_{ij}$$

holds for all $(i, j) \in A$.

Proof Since necessity was proved in Lemma 12.1, we concentrate on the sufficient condition. Suppose $d(j)$ satisfies (12.3), and let $P \equiv s = i_1 - i_2 - \dots - i_k = j$ denote some directed path from s to j . By (12.3), we have

$$\begin{aligned} d(j) &\leq d(i_{k-1}) + c_{i_{k-1}j} \\ &\leq d(i_{k-2}) + c_{i_{k-2}i_{k-1}} + c_{i_{k-1}j} \\ &\quad \vdots \\ &\leq d(i_2) + c_{i_2i_3} + \dots + c_{i_{k-1}j} \\ &\leq d(i_1) + c_{i_1i_2} + \dots + c_{i_{k-1}j} \\ &= \sum_{(i,j) \in P} c_{ij}, \end{aligned}$$

since $d(s) = d(i_1) = 0$. Thus, $d(j)$ is a lower bound on the length of any path from s to j . Since there exists a path of length $d(j)$, this must be the length of the shortest path.

Because (12.3) gives a necessary and sufficient condition for $d(j)$ to be optimal, it is easy to derive a generic algorithm for finding such values, as given in Algorithm 12.1. Note that this algorithm is a local improvement algorithm, since we start with a feasible solution, and that it is guaranteed to end at the optimal solution.

One thing that is missing, though, is that we do not know what the actual shortest paths are. It is reasonable to think that whenever we change the value of $d(j)$, the arc (i, j) will belong to the shortest path to j , assuming that we have a shortest path to i . In fact, we can easily show that this property holds.

Algorithm 12.1 Generic Shortest Path Algorithm

Let $d(j)$ denote the cost of some path from s to j
while there exists an arc $(i, j) \in A$ such that $d(j) > d(i) + c_{ij}$ do $d(j) = d(i) + c_{ij}$.
end while

If the path $s = i_1 - i_2 - \cdots - i_k = j$ is the shortest path from s to j , then $s = i_1 - i_2 - \cdots - i_q$ is the shortest path from s to i_q , where $q = 2, 3, \dots, k - 1$. The arcs of all shortest paths from s to each node comprise a directed tree, where each arc adjacent to s is outgoing, that is, there is no arc (i, s) in this tree.

At each change of $d(j)$, we remove whatever arc goes into j and replace it by (i, j) . This can be done by maintaining a list $\text{pred}(j)$ that gives the tail i of the arc (i, j) . One other change that is often made to the generic algorithm is that, instead of starting with a set of shortest paths, we find the best path at each iteration by assuming at the start that the shortest length $d(j)$ to some large number M for every node j for which a path has yet to be found to j . Hence, we construct our tree of shortest paths one arc at a time, and adjust it as we find better paths. This second algorithm, known as the **Generic Label-Correcting algorithm**, is given in Algorithm 12.2. It is called so because if we view the lengths $d(j)$ as labels for each node, at each iteration we correct those labels that were too large. This is an example of a constructive algorithm that ends with an optimal solution.

Algorithm 12.2 Generic Label-Correcting Algorithm

```

for all  $j \in V$  do
     $d(j) \leftarrow M$                                  $\{M \text{ large number} \implies \text{no path found to } j\}$ 
     $pred(j) \leftarrow -1$                           $\{\text{No predecessor found}\}$ 
end for
 $d(s) \leftarrow 0.$                             $\{\text{Initialize distance of source node}\}$ 
while there exists an arc  $(i, j) \in A$  such that  $d(j) > d(i) + c_{ij}$  do
     $d(j) = d(i) + c_{ij}.$ 
     $pred(j) = i.$ 
end while

```

■ EXAMPLE 12.1

Let's consider the network in [Figure 12.1](#). Our goal is to find the lengths of the shortest paths from $s = 1$ to all other nodes. We assume in the running of the algorithm that if both i and j have $d(i) = d(j) = M$, we will not consider the arc (i, j) . Furthermore, since we do not specify how to choose the arcs, for this example we choose the arc (i, j) with the smallest i that violates (12.2).

We start with node $(1, 2)$ because $d(1) = 0$ and $d(2) = M$. This makes $d(2) = d(1) + c_{12} = 0 + (-2) = -2$. Next, we look at $(1, 4)$, and make $d(4) = 0 + 3 = 3$. The next arc that violates (12.2) is $(2, 4)$, which makes $d(4) = d(2) + (-1) = -3$. Next is $(2, 5)$, which makes $d(5) = -2 + 5 = 3$. Arc $(4, 3)$ violates (12.2), and so we change $d(3)$ to $d(3) = -3 + 1 = -2$. Arc $(3, 5)$ now violates (12.2), so we make $d(5) = -2 + (-2) = -4$. Our next arc to fix is $(4, 6)$, making $d(6) = -3 + 4 = 1$. Next comes arc $(4, 7)$, so $d(7)$ becomes $d(7) = -3 + 2 = -1$. These changes affect the arc $(5, 6)$, so we have to now make $d(6) = -4 + 3 = -1$. Arc $(6, 7)$ violates (12.2), so we make $d(7) = -1 + (-3) = -4$. At this point, it is easy to check that all arcs satisfy (12.2), so these labels contain the lengths of the shortest paths to each node. The following table gives all necessary data, including the paths that yield these lengths.

i	$d(i)$	Path
1	0	1
2	-2	1 - 2
3	-2	1 - 2 - 4 - 3
4	-3	1 - 2 - 4
5	-4	1 - 2 - 4 - 3 - 5
6	-1	1 - 2 - 4 - 3 - 5 - 6
7	-4	1 - 2 - 4 - 3 - 5 - 6 - 7

Now that the generic label-correcting algorithm has been shown to generate

the shortest path and its length from node s to all other nodes, let's consider its convergence properties. Let $C = \max\{|c_{ij}| : (i, j) \in A\}$ be the largest length (in absolute value) over all arcs in G . After any step in which label $d(j)$ has been updated, we have $-(n - 1)C \leq d(j) \leq (n - 1)C$ because any path contains at most $n - 1$ arcs, each with length bounded in absolute value by C . This implies that each node is relabeled at most $2nC$ times, so that the total number of relabeling is bounded by $2n^2C$. Since each iteration performs a relabeling, and it takes at most m steps to determine if a relabeling is needed, we have the following result.

The generic label-correcting algorithm given in Algorithm 12.2 finishes after a finite number of steps to the shortest path distances.

As we did with the minimum spanning tree problem in Chapter 5, once a generic algorithm is identified that takes advantage of the optimality conditions, we see if this algorithm can be modified so that its efficiency improves. Consider the generic label-correcting algorithm. At each iteration, we are asked to identify an arc that violates (12.3). When we find a violated arc (i, j) , we have $d(j) > c_{ij} + d(i)$. At this point in the algorithm, we update (i.e., decrease) $d(j)$ to $d(j) = c_{ij} + d(i)$. How does this change affect other arcs incident to j ? First, consider an arc (k, j) into j . If (k, j) satisfied the optimality condition (12.3) previously, we still have $d(j) \leq c_{kj} + d(k)$. So, all arcs incoming to j that satisfy (12.3) maintain this property. What about arcs (j, k) ? Here, it is possible for an arc that initially satisfied (12.2) to no longer satisfy it, and so when $d(j)$ is changed, we need to look at only the arcs in $A(j)$ to see what has been affected, where $A(j)$ is the set of outgoing arcs from j . This reduces the amount of work we need to do each iteration of the generic label-correcting algorithm.

We can implement this to maintain a list of nodes i whose outgoing arcs (i, k) could *potentially* violate (12.3) due to an update of $d(i)$ during a previous iteration. During each iteration, we add node i to the list if we update $d(i)$. When no more nodes are in this list, we are done. We would start the first iteration with node s , since $d(s)$ is initialized to 0 and all other labels are infinite. This is the approach behind the **Modified Label-Correcting algorithm** given in Algorithm 12.3.

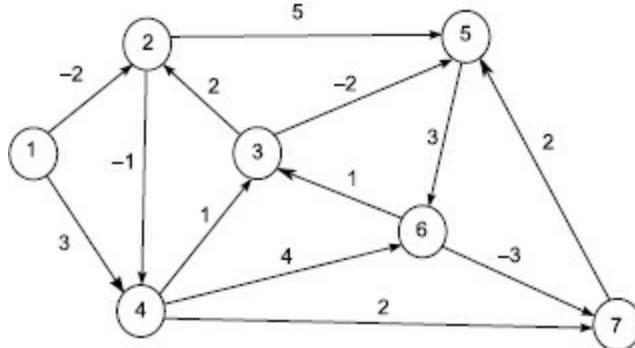
Algorithm 12.3 Modified Label-Correcting Algorithm

```
d(s) ← 0 and pred(s) ← -1.  
d(j) ← M and pred(j) = -1 for each j ∈ V − {s}  
List = {s}.  
repeat  
    Remove i from List  
    for all (i, j) ∈ A(i) do  
        if d(j) > d(i) + cij then  
            d(j) = d(i) + cij.  
            pred(j) = i.  
            if j ∉ List then  
                List ← List ∪ {j}  
            end if  
        end if  
    end for  
until List = ∅
```

■ EXAMPLE 12.2

Let's use the modified label-correcting algorithm to solve the problem given in [Figure 12.2](#). To do this, let's assume that, at each iteration, we remove from *List* the node with smallest index. In the first iteration, we have *List* = {1}, and hence we start with node 1. Scanning the arcs in $A(1)$, we update the values $d(2) \leftarrow -2$, $d(4) \leftarrow 3$, and *List* = {2, 4}. Removing 2 from *List* and scanning $A(2)$, we update $d(4) \leftarrow -3$ and $d(5) \leftarrow 3$, making *List* = {4, 5}. Removing 4 from *List* and scanning $A(4)$, we have $d(3) \leftarrow -2$, $d(6) \leftarrow 1$, $d(7) \leftarrow -1$, with *List* = {3, 5, 6, 7}. We remove 3 from *List* and scan $A(3)$, updating only $d(5) \leftarrow -4$. Next, we remove 5 from *List* and scan $A(5)$, updating only $d(6) \leftarrow -1$. Now, we have *List* = {6, 7}. Removing 6 and scanning $A(6)$ updates only $d(7) \leftarrow -4$. We next remove 7 from *List* but do not update any labels $d(i)$. Since *List* = \emptyset at this point, we are done and all distance labels are optimal.

[FIGURE 12.2](#) Shortest path example.



Algorithm 12.3 is not specific enough on some aspects. For example, it does not say how to choose the next node chosen from *List*. On the positive side, however, the algorithm has some nice flexibility, in that no matter how we choose this next node, we are guaranteed that it will end after a finite number of steps. For example, we could always choose the node from *List* with smallest index. Unfortunately, as given in Exercise 12.5, this could result in a large number of iterations. Another possibility is to choose the node from *List* that has been in *List* the longest. This selection rule follows a “First-In First-Out” (FIFO) policy. To analyze this implementation, we divide its execution into passes where pass 0 consists of the scanning of node s , and pass k consists of the scanning of all nodes that were in *List* at the end of pass $k - 1$. At each pass, we remove a node i from *List* and examine all outgoing arcs (i, j) .

Lemma 12.2 *In this implementation of the modified label-correcting algorithm, at the end of the k th pass, the values of $d(j)$ will be the length of the shortest path for those nodes that have a shortest path requiring at most k arcs.*

A proof can be found in Ahuja et al. [2].

Consider an implementation of the modified label-correcting algorithm that first arranges the arcs in A in some random order and goes through each arc one at a time per pass. Then, at most $n - 1$ iterations through the arcs in A are needed to solve the shortest path problem.

This approach of first ordering the arcs and proceeding through the list of arcs in the same order each iteration is known as the **FIFO label-correcting algorithm**.

■ EXAMPLE 12.3

Let's use the FIFO label-correcting algorithm to solve the problem given in [Figure 12.2](#). In the first iteration, we start with $List = \{1\}$, and hence we start with node 1. Scanning the arcs in $A(1)$, we update the values $d(2) \leftarrow -2$, $d(4) \leftarrow 3$, and $List = \{2, 4\}$. Removing 2 from $List$ and scanning $A(2)$, we update $d(4) \leftarrow -3$ and $d(5) \leftarrow 3$, making $List = \{4, 5\}$. Removing 4 from $List$ and scanning $A(4)$, we have $d(3) \leftarrow -2$, $d(6) \leftarrow 1$, $d(7) \leftarrow -1$, with $List = \{5, 3, 6, 7\}$. We remove 5 from $List$ and scan $A(5)$, updating nothing. Next, we remove 3 from $List$ and scan $A(3)$, updating only $d(5) \leftarrow -4$. Now, we have $List = \{6, 7, 5\}$. Removing 6 and scanning $A(6)$ update only $d(7) \leftarrow -2$. We next remove 7 from $List$, but do not update any label $d(i)$. Next, we remove 5 from $List$, scan $A(5)$, and update only $d(6) \leftarrow -1$. Now, $List = \{6\}$. Removing 6, we now update $d(7) \leftarrow -4$ and $List = \{7\}$. Removing 7 from $List$ allows us to update nothing. Since $List = \emptyset$ at this point, we are done and all distance labels are correct.

Shortest Paths with Nonnegative Lengths Let's now consider the case where the lengths c_{ij} are nonnegative. Suppose that at each iteration of the modified label-correcting algorithm, we choose the node j whose distance label $d(j)$ is the minimum over all nodes in $List$. We can assume that its predecessor $pred(j)$ is not in $List$ (why?). Hence, $pred(j)$ must have been in $List$ at some point and then removed. It seems reasonable that when a node is removed from $List$, its distance label is in fact the length of the shortest path to j . The following indicates this to be correct.

At a given iteration, we remove a node j from $List$ if its distance label $d(j)$ is minimum over all distance labels $d(k)$, where $k \in List$. Let S denote the nodes that have been removed from $List$ at some previous iteration. Then

1. for all $i \in S$, the distance labels $d(i)$ are the lengths of the shortest path from s to i , and
2. for all $i \notin S$, the distance labels $d(i)$ are the lengths of the shortest path from s to i using only nodes in S as intermediate nodes.

If at each iteration of the modified label-correcting algorithm, we remove the node j from $List$ whose label $d(j)$ is minimum over all other nodes in

List, then node j cannot be placed into *List* at any subsequent iteration.

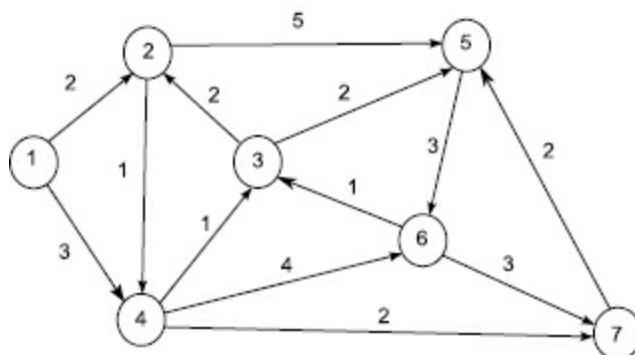
Since each node appears in *List* at most once, each arc (i, j) is examined only once. This leads to a special implementation of the modified label-correcting algorithm known as **Dijkstra's Algorithm**, which is given formally in Algorithm 12.4.

Algorithm 12.4 Dijkstra's Algorithm

```

 $d(s) \leftarrow 0$  and  $pred(s) \leftarrow -1$ .
 $d(j) \leftarrow M$  and  $pred(j) \leftarrow -1$  for each  $j \in V - \{s\}$ 
List =  $\{s\}$ 
repeat
    Remove  $i$  from List such that  $d(i) = \min\{d(k) : k \in \text{List}\}$ .
    for all  $(i, j) \in A(i)$  do
        if  $d(j) > d(i) + c_{ij}$  then
             $d(j) = d(i) + c_{ij}$ .
             $pred(j) = i$ .
        if  $j \notin \text{List}$  then
            List  $\leftarrow \text{List} \cup \{j\}$ 
        end if
    end if
    end for
until List =  $\emptyset$ 
```

FIGURE 12.3 Shortest path problem with nonnegative costs.



■ EXAMPLE 12.4

Consider the example given in [Figure 12.3](#). Our first iteration starts with $j =$

1, and so we update $d(2) \leftarrow 2$, $d(4) \leftarrow 3$, making $List = \{2, 4\}$. Since node 2 has smallest label, it is removed and $A(2)$ is scanned, updating $d(5) \leftarrow 7$. $List$ is now $\{4, 5\}$. Node 4 is now removed, so that we relabel $d(3) \leftarrow 4$, $d(6) \leftarrow 7$, $d(7) \leftarrow 5$ and $List = \{3, 5, 6, 7\}$. Node 3 has smallest label, so it is removed and $A(3)$ is scanned. This changes $d(5) \leftarrow 6$, and $List = \{5, 6, 7\}$. Since node 7 has smallest label, it is removed, but we update no labels. Next, node 5 is removed, and again we update no labels. Finally, node 6 is removed. Since no labels are again updated, and $List = \emptyset$, our algorithm stops, and all distance labels are correct.

12.3 MAXIMUM FLOW PROBLEMS

Many fundamental combinatorial optimization problems center around the concept of “flow.” We saw this in Chapter 2 when we considered various network models. Now that we’ve considered the minimum spanning tree problem (in Chapter 5) and the shortest path problem, it’s now time to look at flow problems. The previous problems have both primarily concerned themselves with finding paths and trees on a network. While it is true the shortest path problem can be viewed as a special case of the minimum cost network flow problem, we really do not use this fact. In this section, we are interested in determining the maximum amount of flow we can send in a network.

Maximum Flow Problem Given a network $G = (V, A)$ with nonnegative arc capacities $u_{ij} \geq 0$ for each $(i, j) \in A$, the *Maximum Flow Problem* is to determine the maximum amount of flow that can be sent from a designated node s to another designated node t without violating any of the arc capacities.

Node s , where the flow originates, is known as the **source node**, while node t , where the flow ends, is known as the **sink node**.

We can state the maximum flow problem as the following linear program:

$$\begin{aligned} \max \quad & v \\ (12.4a) \quad \text{s.t.} \quad & \end{aligned}$$

$$(12.4b) \quad \sum_{j:(i,j) \in A} x_{ij} - \sum_{k:(k,i) \in A} x_{ki} = \begin{cases} v, & \text{if } i = s, \\ -v, & \text{if } i = t, \\ 0, & \text{otherwise,} \end{cases} \quad i \in V$$

$$(12.4c) \quad 0 \leq x_{ij} \leq u_{ij}, \quad (i, j) \in A.$$

We shall call the vector \mathbf{x} that satisfies (12.4b) and (12.4c) a *flow* and v will denote the *value of the flow*.

Before we begin attempting to solve this problem, let's first make some basic assumptions about the problem and the network.

Assumptions

1. The network $G = (V, A)$ is a directed network.
2. For every $i, j \in V$, we have $(i, j) \in A$ and $(j, i) \in A$. Note that this is not very restrictive, since we indicate only those arcs with $u_{ij} > 0$.
3. There are no parallel arcs between nodes i and j .
4. All arc capacities u_{ij} are nonnegative integers or $u_{ij} = \infty$.
5. There does not exist a path from s to t where each arc on the path has infinite capacity. This assumption only means that there is a finite solution to our problem.

When is a Given Flow Optimal? If we are given a flow \mathbf{x} on a network G , how can we determine if it is a maximum flow? As we've seen previously, the concept of an optimality condition is often key to finding algorithms for optimization problems. Unfortunately, unlike for the minimum spanning tree and shortest path problems, optimality conditions are not as obvious for maximum flow problems.

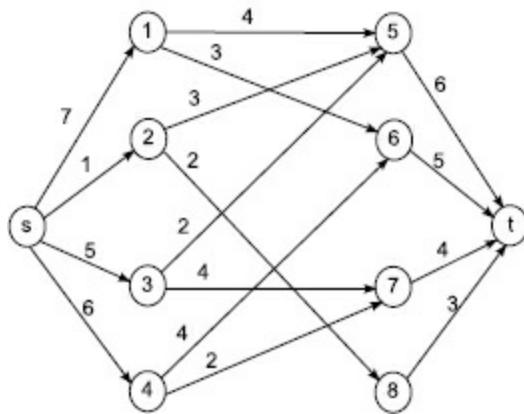
Suppose we are given the network found in [Figure 12.4](#). To find the maximum flow from s to t , one natural way to start is to just send as much flow as possible along paths from s to t . If we were to send flow along paths on which we can send the most flow, we could send flow along these paths (in the case of ties, we assume that the first node following s had smallest index)



Path	Flow Amount
s-1-5-t	4
s-3-7-t	4
s-4-6-t	4
s-1-6-t	1
s-2-5-t	1
s-3-5-t	1

giving us a flow of value 15. Is this optimal?

FIGURE 12.4 Maximum flow example problem.



One way to see if this is the optimal flow value is to find an upper bound to the maximum flow value, similar to our thinking in Chapter 9 when we first examined linear programming duality. If we find an upper bound to the maximum flow value, and our flow has this value, then we must have the maximum flow. For our network, note that the maximum flow value must be no more than 19, since this is the sum of all arc capacities leaving s . Closer inspection shows that a better upper bound is 17, or the sum of all arc capacities entering t . This does not mean that there is a flow of 17, just that our maximum flow cannot have a value greater than 17. So, now the question is, is this the best upper bound we can find?

If you look closely at how we found these upper bounds, you may notice that we partitioned the nodes into two groups S and \bar{S} , where $s \in S$ and $t \in \bar{S}$. We then added up the capacities of all arcs (i, j) in which $i \in S$ and $j \in \bar{S}$. Formalizing this, we have the following definitions.

s-t cut Given a network $G = (V, A)$, an $s-t$ cut (S, \bar{S}) is a partition of the

nodes

$V = S \cup \bar{S}$ ($S \cap \bar{S} = \emptyset$) with $s \in S$ and $t \in \bar{S}$.

The **capacity of an s-t cut** (S, \bar{S}) , denoted $u[S, \bar{S}]$, is defined as

$$u[S, \bar{S}] = \sum_{(i,j) \in (S, \bar{S})} u_{ij},$$

where $(i, j) \in (S, \bar{S})$ means that $i \in S$ and $j \in \bar{S}$.

Given a flow \mathbf{x} of value v , the amount of flow that crosses the s-t cut (S, \bar{S}) is

$$v = v(S, \bar{S}) = \sum_{(i,j) \in (S, \bar{S})} x_{ij} - \sum_{(j,i) \in (\bar{S}, S)} x_{ji},$$

or the amount of flow going from S to \bar{S} minus the amount of flow going from \bar{S} to S . Based upon this definition, and our earlier observation on upper bounds, we have the following lemma.

Lemma 12.3 *Let v denote the value of any flow. Then,*

$$v \leq u[S, \bar{S}]$$

for all s-t cuts (S, \bar{S}) .

Proof Let \mathbf{x} be a flow with value v , and let (S, \bar{S}) be any s-t cut. Then

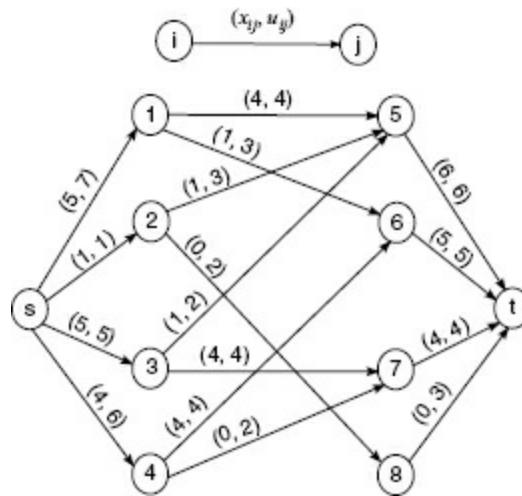
$$\begin{aligned} v &= v(S, \bar{S}) \\ &= \sum_{(i,j) \in (S, \bar{S})} x_{ij} - \sum_{(j,i) \in (\bar{S}, S)} x_{ji} \\ &\leq \sum_{(i,j) \in (S, \bar{S})} u_{ij} - \sum_{(j,i) \in (\bar{S}, S)} 0 \\ &= u[S, \bar{S}]. \end{aligned}$$

Lemma 12.3 looks like a weak duality result! We have seen before that if we have a flow \mathbf{x} that has the same value as the capacity of any s-t cut, then \mathbf{x} is a maximum flow. Hence, we need to decide if this criterion is ever satisfied. If we look closely, we see that the cut (S, \bar{S}) , where $S = \{s, 1, 3, 4, 6, 7\}$, has a capacity of 16. Hence, our current flow, if not optimal, is very close to optimal.

Our next step is to see if there is a flow of value 16 or an s-t cut with capacity 15. Looking for a cut of a given capacity is nontrivial, so perhaps we should try to see if our flow can be improved upon. Consider [Figure 12.5](#), which gives our current solution with value 15. Notice that the arc $(8, t)$ can

still carry flow on it, as well as the arc $(s, 4)$. Unfortunately, there is no arc $(4, 8)$ to send any flow. However, closer inspection shows that if we could send the flow that is on $(2, 5)$ and $(5, t)$ along the arcs $(2, 8)$ and $(8, t)$, then we have another flow of value 15. However, this change frees up the arc $(5, t)$ by one unit of flow, which we could use by sending only three units of flow from node 3 to node 7 and sending two units from node 3 to node 5. Here is another flow of value 15. This change now frees up some flow along the arc $(7, t)$, which we can use to send flow along the path $s - 4 - 7 - t$. Since this is a new unit of flow, we now have a flow with value 16, which we know must be a maximum flow, since we have an s - t cut with capacity 16. Summarizing all the changes, we increased the flow along arcs $(s, 4)$ and $(4, 7)$ by one unit, decreased the flow along $(3, 7)$ by one unit, increased the flow along $(3, 5)$ by one unit, decreased the flow of $(2, 5)$ by one unit, and sent another unit of flow along the arcs $(2, 8)$ and $(8, t)$.

FIGURE 12.5 Maximum flow of value 15.



Here, we've seen that, for this network, we have a flow whose value equals the value of some s - t cut, and we found this flow by either increasing or decreasing the amount of flow sent along arcs in some “undirected” path from s to t . As problem solvers and operations researchers, we should ask ourselves the following questions: (1) does every network have a flow and an s - t cut whose value/capacity is equal, thus giving us an optimal flow, and (2) can we generalize and always use this trick of increasing or decreasing flow on arcs to find another flow of larger value? We shall answer both of these questions, beginning with the latter.

Residual Networks Given a flow \mathbf{x} , for each $(i, j) \in A$, we can increase the net amount of flow traveling from i to j by either (1) increasing the flow x_{ij} sent from i to j or (2) decreasing the flow x_{ji} sent from j to i . How much can we increase this flow? Depending on which method we use, we could either increase x_{ij} by $u_{ij} - x_{ij}$ or decrease x_{ji} by x_{ji} . So, define

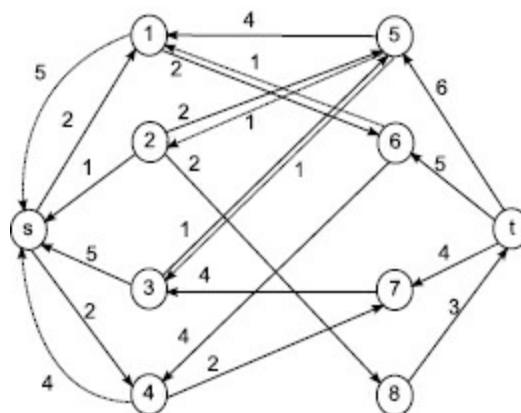
$$r_{ij} = u_{ij} - x_{ij} + x_{ji}$$

to be the total amount of increase possible in the amount of flow from i to j along one arc. We shall call r_{ij} the **residual capacity** of (i, j) . Then, for any flow \mathbf{x} , we can define the **Residual Network** $G(\mathbf{x}) = (V, A_x)$, where, for every $(i, j) \in A$, there exists an arc $(i, j) \in G(\mathbf{x})$ with capacity $u_{ij} - x_{ij}$ and an arc (j, i) with capacity x_{ji} . If parallel arcs exist in $G(\mathbf{x})$, we can simply combine them into one arc whose capacity is the sum of the capacities of all parallel arcs. The residual network then gives us information on how we can change the flow along all arcs. For example, [Figure 12.6](#) contains the residual network for the flow of value 15 that we first looked at. In this residual network, there is a path $P = s-4-7-3-5-2-8-t$, with $r_{ij} > 0$ for each $(i, j) \in P$, along which we can send one unit of flow from s to t .

In general, let $P = s - i_1 - i_2 - \dots - i_k - t$ be a directed path from s to t in G (where $r_{ij} > 0$ for each $(i, j) \in P$). We call path P an **Augmenting Path**, and we can then send

$$\delta = \min\{r_{ij} : (i, j) \in P\}$$

[FIGURE 12.6](#) Residual flow example.



units of flow along P , which generates a flow of larger value. If $r_{ij} = \delta$ for

some arc $(i, j) \in P$, we say that arc (i, j) has been *saturated*.

Given a feasible flow \mathbf{x} , if there exists an augmenting path P in $G(\mathbf{x})$, then we can find another flow \mathbf{x}' whose value is greater than that of \mathbf{x} .

How much can we increase the value of \mathbf{x} by sending flow along augmenting paths? Suppose \mathbf{x} is a flow of value v and we want to find a flow whose value is $v + \Delta v$, where $\Delta v \geq 0$. Given an s-t cut (S, \bar{S}) , we know that

$$v + \Delta v \leq u[S, \bar{S}] = \sum_{i \in S, j \in \bar{S}} u_{ij}$$

and that

$$\begin{aligned} v &= \sum_{i \in S, j \in \bar{S}} x_{ij} - \sum_{i \in \bar{S}, j \in S} x_{ij} \\ &= \sum_{i \in S, j \in \bar{S}} x_{ij} - \sum_{i \in S, j \in \bar{S}} x_{ji}. \end{aligned}$$

Hence,

$$\begin{aligned} \Delta v &\leq \sum_{i \in S, j \in \bar{S}} (u_{ij} - x_{ij}) + \sum_{i \in S, j \in \bar{S}} x_{ji} \\ &= \sum_{i \in S, j \in \bar{S}} r_{ij}. \end{aligned}$$

For any flow \mathbf{x} of value v , the amount of additional flow Δv that can be sent from s to t satisfies

$$\Delta v \leq \sum_{i \in S, j \in \bar{S}} r_{ij} = r[S, \bar{S}]$$

for all s-t cuts (S, \bar{S}) in $G(\mathbf{x})$.

This implies that if there exists a cut (S, \bar{S}) in $G(\mathbf{x})$ with $r[S, \bar{S}] = 0$, then we cannot send any additional flow. Does this make the flow \mathbf{x} a maximum flow? The following theorem says yes.

Theorem 12.1 A flow \mathbf{x} is a maximum flow if and only if there is an s-t cut (S, \bar{S}) in $G(\mathbf{x})$ with $r[S, \bar{S}] = 0$.

Proof Suppose that \mathbf{x} is a flow of maximum value. By above, there does not exist an augmenting path in $G(\mathbf{x})$. Let S denote the nodes that are reachable from s via a path along arcs in $G(\mathbf{x})$ with $r_{ij} > 0$, and let $\bar{S} = V - S$. Let $i \in S$

and $j \in \bar{S}$. If $u_{ij} > 0$, then we must have $x_{ij} = u_{ij}$ and $x_{ji} = 0$, else we would have $r_{ij} > 0$ and j would be reachable from s . If $u_{ij} = 0$, then we have $x_{ji} = 0$, else again, j would be reachable from s . Hence, $r_{ij} = 0$ and thus $r[S, \bar{S}] = 0$.

Now, suppose that there is an s-t cut (S, \bar{S}) in $G(\mathbf{x})$ with $r[S, \bar{S}] = 0$. Since each $(i, j) \in (S, \bar{S})$ has $r_{ij} = 0$, and $r_{ij} = (u_{ij} - x_{ij}) + x_{ji}$, this must mean that $x_{ij} = u_{ij}$ and $x_{ji} = 0$ since both components of r_{ij} are nonnegative. Since the value of the flow \mathbf{x} is

$$v = \sum_{i \in S, j \in \bar{S}} x_{ij} - \sum_{i \in S, j \in \bar{S}} x_{ji} = \sum_{i \in S, j \in \bar{S}} u_{ij},$$

$$u[S, \bar{S}] = \sum_{i \in S, j \in \bar{S}} u_{ij},$$

and the capacity of (S, \bar{S}) in G is $u[S, \bar{S}]$ we have a flow and a cut whose value and capacity are equal. Thus, our flow must be a maximum flow.

Corollary 12.1 *A flow \mathbf{x} is a maximum flow if and only if there is no augmenting path in $G(\mathbf{x})$ from s to t .*

We also get one of the fundamental results in combinatorial optimization.

Max-Flow Min-Cut Theorem A flow \mathbf{x} is a maximum flow of value v if and only if there exists an s-t cut (S, \bar{S}) in G with $u[S, \bar{S}] = v$. The value $u[S, \bar{S}]$ is the minimum capacity of all s-t cuts in G .

Let \mathbf{x} be a maximum flow with value v . The s-t cut (S, \bar{S}) can be found by setting S to be those nodes that are reachable from s via a path along arcs in $G(\mathbf{x})$ with $r_{ij} > 0$, and setting $\bar{S} = V - S$.

We have two seemingly unrelated problems where each problem generates a bound (upper or lower) for the other and whose optimal solution values are the same. Such a result is a *Strong Duality Result*. It is also easy to show that the Max-Flow Min-Cut Theorem is a special case of linear programming duality (see Exercise 12.18).

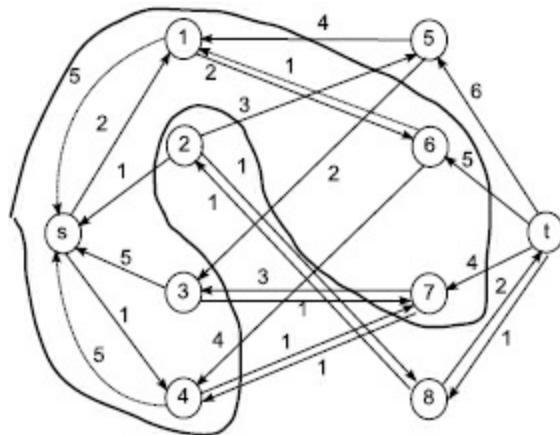
■ EXAMPLE 12.5

We earlier identified a flow of value 16 in the network given in [Figure 12.4](#). The residual network for this flow is given in [Figure 12.7](#), which also includes the cut $(S, \bar{S}) = (\{s, 1, 3, 4, 6, 7\}, \{2, 5, 8, t\})$. Note that the nodes in

S are precisely those that are reachable from s in this residual network. The capacity of this cut is

$$\sum_{i \in S, j \in \bar{S}} u_{ij} = u_{s2} + u_{15} + u_{35} + u_{6t} + u_{7t} = 16.$$

FIGURE 12.7 Residual network of optimal flow with minimum cut.



It is time to form an algorithm for solving the maximum flow problem. It hopefully is apparent that we can use Corollary 12.1 to generate an algorithm, namely, search for an augmenting path. If one exists, send flow along it, and if one does not exist, then we have an optimal solution. This algorithm, given in Algorithm 12.5, is called the **Generic Augmenting Path Algorithm**.

Algorithm 12.5 Generic Augmenting Path Algorithm

```

 $x = 0.$  {Zero initial flow}
while  $G(x)$  has an augmenting path from  $s$  to  $t$  do
    Identify an augmenting path  $P$  from  $s$  to  $t$ 
    Augment  $\delta = \min\{r_{ij} : (i, j) \in P\}$  units of flow along  $P$ 
    Update  $x, G(x)$ 
end while

```

While this is a very simple algorithm, it is perhaps too simple and generic. For example, it does not tell us how to find an augmenting path, and it is not apparent that the algorithm ends in a finite number of steps. However, a number of algorithms for solving the maximum flow problem are really specific implementations of the generic augmenting path algorithm.

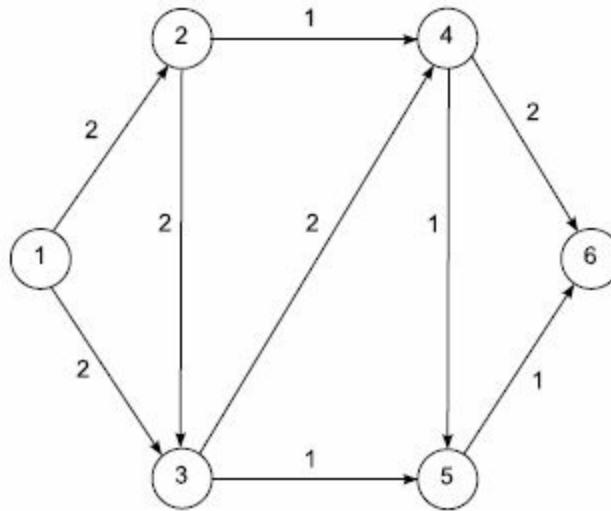
One such algorithm is the **Labeling Algorithm** introduced by Ford and Fulkerson in 1956 and given in Algorithm 12.6. They used a graph search to find an augmenting path and used Theorem 12.1 to identify whether an

augmenting path existed or not. Note that this approach maintains predecessor nodes similar to those used in the shortest path problem. Here, if t is labeled, an augmenting path has been found.

■ EXAMPLE 12.6

Consider the network given in [Figure 12.8](#), with $s = 1$ and $t = 6$, where the values on the arcs are the capacities. In the first iteration, we start with $List = \{1\}$.

[FIGURE 12.8](#) Network for Example 12.6.



Algorithm 12.6 Labeling Algorithm

```

Label node  $t$ .
 $\mathbf{x} = \mathbf{0}$ . { Zero initial flow }
while  $t$  is labeled do
    Unlabel all nodes
    Set  $\text{pred}(j) = 0 \forall j \in V$ 
    Label  $s$ ;  $List := \{s\}$ 
    while  $List \neq \emptyset$  and  $t$  unlabeled do
        Remove  $i$  from  $List$ 
        for all  $(i, j) \in G(\mathbf{x})$  do
            if  $r_{ij} > 0$  and  $j$  unlabeled then
                Label  $j$ 
                 $\text{pred}(j) = i$ 
                Add  $j$  to  $List$ 
            end if
        end for
    end while
    if  $t$  is labeled then
        Use  $\text{pred}(\cdot)$  labels to identify augmenting path  $P$ 
         $\delta = \min \{r_{ij} : (i, j) \in P\}$ 
        augment  $\delta$  units of flow along  $P$ 
        update  $\mathbf{x}$ ,  $G(\mathbf{x})$ 
    end if
end while

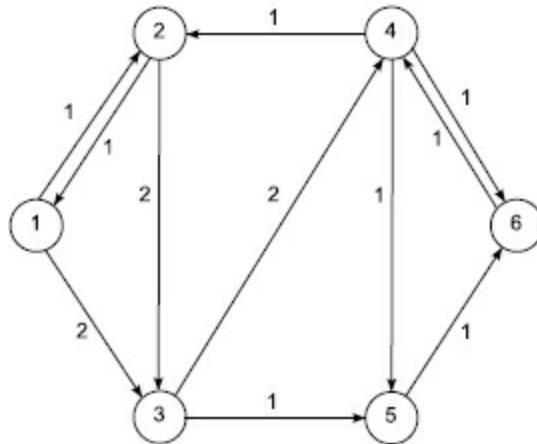
```

Removing node 1 from $List$, we examine $(1, j) \in G(\mathbf{x})$ and label nodes 2, 3, so that $List = \{2, 3\}$. We next choose to remove node 2 from $List$. Examining $(2, j) \in G(\mathbf{x})$, we label node 4, yielding $List = \{3, 4\}$. Removing node 4 from $List$, we examine $(4, j) \in G(\mathbf{x})$, we label nodes 5 and 6, giving $List = \{3, 5, 6\}$. Removing node 6 from $List$, we have found an augmenting path from node 1 to node 6, namely, $P_1 = 1-2-4-6$. Since $1 = \min\{r_{ij} : (i, j) \in P_1\}$, we augment one unit of flow along P_1 . The new residual graph $G(\mathbf{x})$ is given in [Figure 12.9](#). In the second iteration, we again start with $List = \{1\}$. Removing node 1 from $List$, we next get $List = \{2, 3\}$. If we remove node 2 from $List$, there are no arcs $(2, j) \in G(\mathbf{x})$ with $r_{2j} > 0$ and j is unlabeled; hence, we label no new nodes. Next, we remove node 3 from $List$ and thus label nodes 4, 5, so that $List = \{4, 5\}$. Removing node 4 from $List$, we label node 6 and add it to $List$. If we remove node 6 from $List$, we have found a second augmenting path from 1 to 6, namely, $P_2 = 1-3-4-6$. Since $1 = \min\{r_{ij} : (i, j) \in P_2\}$ (since $r_{46} = 1$), we augment one unit of flow along P_2 . The new residual graph $G(\mathbf{x})$ is given in [Figure 12.10](#).

In the third iteration, we again start with $List = \{1\}$. Removing node 1 from $List$, we next get $List = \{2, 3\}$. If we remove node 2 from $List$, there are no arcs $(2, j) \in G(\mathbf{x})$ with $r_{2j} > 0$ and j is unlabeled; hence, we label no new nodes. Next, we remove node 3 from $List$ and thus label nodes 4, 5, so that

$List = \{4, 5\}$. Removing node 4 from $List$, we have no arcs $(4, j)$ with $r_{4j} > 0$ and j unlabeled, thus we do nothing. If we remove node 5 from $List$, we add node 6 to $List$. Removing node 6 from $List$, we have found a third augmenting path from 1 to 6, namely, $P_3 = 1-3-5-6$. Since $1 = \min\{r_{ij} : (i, j) \in P_3\}$ (since $r_{35} = 1$), we augment one unit of flow along P_3 . The new residual graph $G(\mathbf{x})$ is given in [Figure 12.11](#).

FIGURE 12.9 First residual network for Example 12.6.



In the fourth iteration, we again start with $List = \{1\}$. Removing node 1 from $List$, we next get $List = \{2\}$, since $r_{13} = 0$. If we remove node 2 from $List$, we label node 3, so $List = \{3\}$. Removing node 3 from $List$, we label node 4, making $List = \{4\}$. Removing node 4 from $List$, we can label node 5, so $List = \{5\}$. Removing node 5 from $List$, we find no arcs $(5,j) \in G(\mathbf{x})$ with $r_{5j} > 0$ and j unlabeled. Hence, we are done. The maximum flow \mathbf{x} has value 3, and a minimum s-t cut is $(\{1, 2, 3, 4, 5\}, \{6\})$ with capacity 3. The maximum flow is given in [Figure 12.12](#).

FIGURE 12.10 Second residual network for Example 12.6.

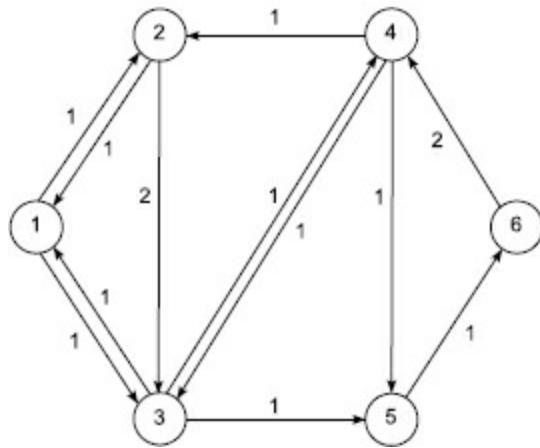


FIGURE 12.11 Third residual network for Example 12.6.

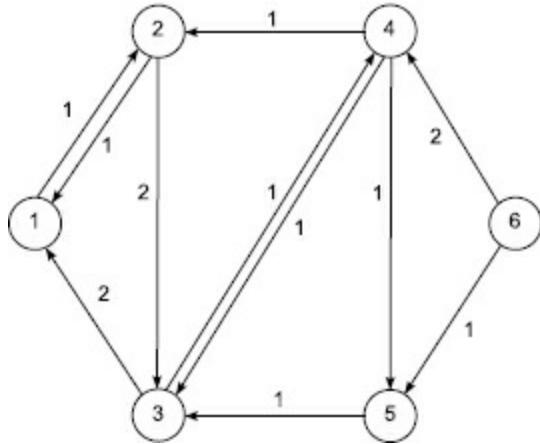
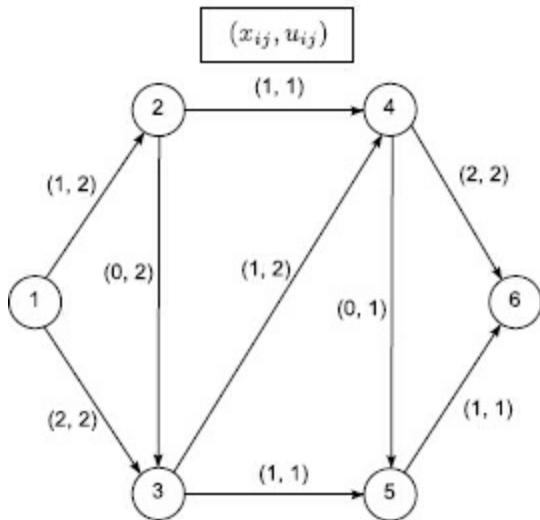


FIGURE 12.12 Optimal maximum flow for Example 12.6.



It is easy to see that the labeling algorithm is correct because of Theorem

12.1. One immediate consequence of the labeling algorithm is the following characterization of optimal flows.

If all capacities u_{ij} are integers, then the maximum flow has an integer value and there exists a maximum flow \mathbf{x} such that x_{ij} is an integer.

The labeling algorithm does have some drawbacks. It can be shown (see Ahuja et al. [2]) that if the capacities u_{ij} are irrational, the algorithm may not terminate, and if it does, it may converge to a nonoptimal solution. In addition, you may have noticed that the algorithm “forgets” what it did during the last augmentation, even though only a small portion of the network changes during each loop. For ways to modify the labeling algorithm to fix these issues, the book by Ahuja, Magnanti, and Orlin [2] is an excellent resource.

12.4 MINIMUM COST NETWORK FLOW PROBLEMS

In previous sections, we have looked at ways to determine a path from one node to another with minimum cost, as well as how to send the most flow from a designated source node to a designated sink node. In both problems, we were interested in sending “stuff” from one locale to another; one problem was concerned with minimum cost, the other with sending as much as possible. Both of these problems ignore important components of the other. In this section, we look to combine them: send a predetermined amount of flow from various nodes to others at minimum cost. This problem is known as the *Minimum Cost Network Flow Problem (MCNF)*.

Minimum Cost Network Flow Problem Let $G = (V, A)$ be a directed network with both a cost c_{ij} and a capacity $u_{ij} \geq 0$ associated with each $(i, j) \in A$. In addition, to each node $i \in V$ we will associate a number $b(i)$ that indicates the supply (if $b(i) > 0$) or demand (if $b(i) < 0$) of flow to node i . The *Minimum Cost Network Flow Problem* is to send flow at minimum cost while meeting arc capacities and, at each node i , the net

flow through i is equal to $b(i)$.

We can formulate the minimum cost network flow problem as the linear program

$$(12.5a) \quad \min \sum_{(i,j) \in A} c_{ij}x_{ij}$$

s.t.

$$(12.5b) \quad \sum_{j:(i,j) \in A} x_{ij} - \sum_{k:(k,i) \in A} x_{ki} = b(i), \quad i \in V$$

$$(12.5c) \quad 0 \leq x_{ij} \leq u_{ij}, \quad (i, j) \in A,$$

where x_{ij} represents the flow on the arc (i, j) . A vector \mathbf{x} that satisfies (12.5b) and (12.5c) will be called a **feasible flow** or **flow**.

We shall make the following assumptions concerning our network.

Assumptions

1. All data $(c_{ij}, u_{ij}, b(i))$ are integral.
2. The network is directed and connected.
3. The supply/demand values $b(i)$ satisfy

$$\sum_{i \in V} b(i) = 0.$$

Furthermore, MCNF has a feasible solution. *Note:* We can check this assumption by solving a maximum flow problem; see Exercise 12.19.

4. All arc costs $c_{ij} \geq 0$.
5. There are no parallel arcs in G .
6. If arc (i, j) has $u_{ij} > 0$, then arc (j, i) must have $u_{ji} = 0$ and $c_{ji} = 0$.

Since we are talking about flows over a network, we need to redefine a *residual network* to incorporate the costs c_{ij} . Given a flow \mathbf{x} , the residual network $G(\mathbf{x})$ contains arc $(i, j) \in A$ with cost c_{ij} and capacity $r_{ij} = u_{ij} - x_{ij} + x_{ji}$ and an arc (j, i) with cost $c_{ji} = -c_{ij}$ and capacity $r_{ji} = x_{ij}$

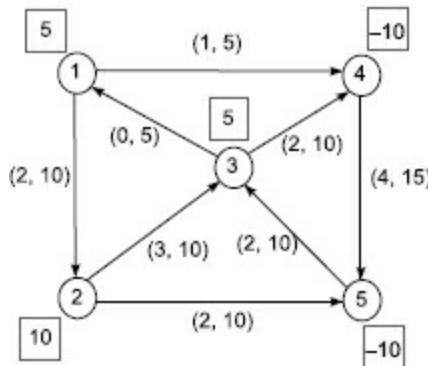
Cycle-Canceling Algorithm As we have with the previous problems in this chapter, we begin by first determining an optimality condition for the problem by using an example problem and then building an algorithm around

this condition.

Consider the network given in [Figure 12.13](#), where the boxed numbers are the values of $b(i)$ and the numbered pair on arc (i, j) are (c_{ij}, u_{ij}) . [Figure 12.14](#) gives its residual network, corresponding to the feasible flow $x_{14} = x_{23} = x_{25} = x_{45} = 5, x_{34} =$

10. In the residual network, consider the directed cycle $W = 1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 1$. We can augment flow around this cycle, as each unit of flow changes the overall cost of the flow by $\Delta = 2 + 2 - 4 - 2 + 0 = -2$. Hence, by sending flow around this negative cycle, we can reduce the cost of a feasible flow. It is easy to see that if the residual network $G(\mathbf{x})$ of a feasible flow \mathbf{x} has a negative cost directed cycle, then \mathbf{x} cannot be an optimal solution. The following optimality condition indicates that the converse is also true.

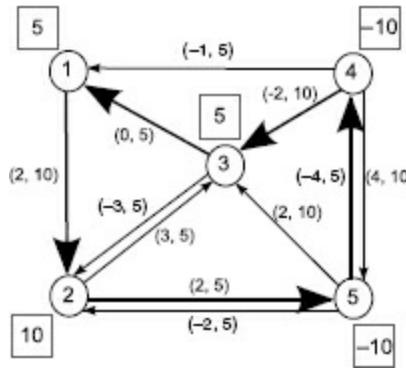
[FIGURE 12.13](#) Minimum cost flow example network.



Negative Cycle Optimality Condition A feasible flow \mathbf{x}^* is an optimal solution of the minimum cost flow problem if and only if its residual network $G(\mathbf{x}^*)$ contains no negative cost directed cycle.

A proof of this can be found in Ahuja et al. [2]. This optimality condition automatically implies a generic algorithm for solving the minimum cost network flow problem: Given a feasible flow \mathbf{x} , determine whether there is a negative cycle in $G(\mathbf{x})$. If yes, augment flow around that cycle and update $G(\mathbf{x})$; if no, we have an optimal flow. This is the **Cycle-Canceling Algorithm** and is given in Algorithm 12.7.

[FIGURE 12.14](#) Residual network with negative cycle.



Algorithm 12.7 Cycle Canceling Algorithm

```

Establish a feasible flow  $\mathbf{x}$ .
while  $G(\mathbf{x})$  contains a negative cost cycle do
    Identify a negative cycle  $W$ .
     $\delta = \min\{r_{ij} : (i, j) \in W\}$ .
    Push  $\delta$  units of flow along  $W$ .
    Update  $G(\mathbf{x})$ .
end while

```

■ EXAMPLE 12.7

Consider again the following minimum cost network flow problem given in [Figure 12.13](#), where the boxed numbers are the values of $b(i)$ and the numbered pair on arc (i, j) are (c_{ij}, u_{ij}) . Consider the following feasible flow on this network:

$$x_{14} = x_{23} = x_{25} = x_{45} = 5, x_{34} = 10.$$

Figure 12.14 shows the resulting residual network. Earlier we saw the negative cycle $1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 1$. We can push up to $\delta = 5$ units around this cycle, which generates the residual network in [Figure 12.15](#). Examining this residual network, we find a negative cycle $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$. Pushing $\delta = 5$ units around this cycle produces the residual network in [Figure 12.16](#). Since this residual network contains no negative cycles, the current flow

$$x_{14} = x_{34} = 5, x_{25} = 10$$

is optimal.

[FIGURE 12.15](#) Second negative cycle.

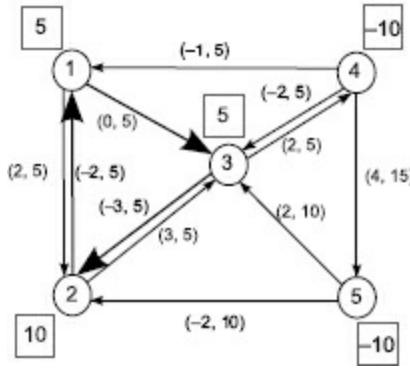
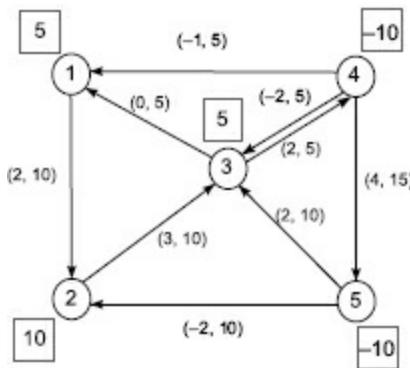


FIGURE 12.16 Optimal residual network.



A sticking point of this algorithm is finding a negative cycle in $G(\mathbf{x})$ or determining that none exists. In Exercise 12.6, we show that this can be done using the FIFO label-correcting algorithm and the predecessor arcs $(\text{pred}(j), j)$.

One final note about this algorithm. Much like the labeling algorithm for solving maximum flow problems, we augment flow by augmenting the most flow around each cycle, and this amount is the maximum of all residual capacities on this cycle. Since each residual capacity changes by subtraction (or addition) from one iteration to the next, we get the following result characterizing solutions.

If all arc capacities u_{ij} and all node values $b(i)$ are integers, then there exists a solution to the minimum cost flow problem with integral arc flows, that is, x_{ij} is an integer for all $(i, j) \in A$.

Network Simplex Algorithm

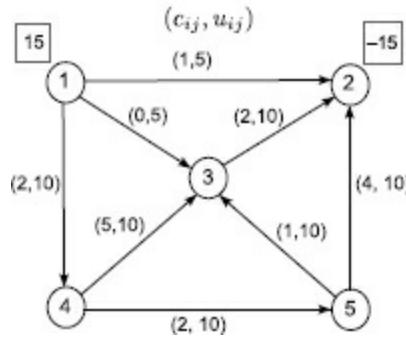
In Chapter 11.2, we derived the transportation simplex method, a variant of

the simplex method that took advantage of various properties of the transportation problem. Since the minimum cost network flow problem (12.5) has upper bounds on all variables, we could solve this linear program using the bounded simplex method. A natural question is whether we can fine-tune the bounded simplex method for (12.5).

Recall that the bounded simplex method requires that we be able to answer the following questions:

1. What is the structure of an extended basic (feasible) solution to our problem?
2. How do we calculate reduced costs and determine the entering variable?
3. How do we determine the leaving variable, and hence our new extended basic feasible solution?

FIGURE 12.17 Example problem for network simplex algorithm.



For the transportation problem, we saw that a basic solution corresponded to a spanning tree on the network (see Lemma 11.1). Is this still true for the general minimum cost flow problem? To answer this, note that we assumed that

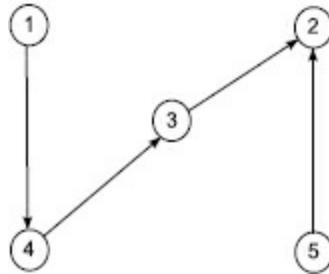
$$\sum_{i \in V} b(i) = 0$$

and that there was a feasible solution to (12.5). Since each arc $(i, j) \in A$ is in exactly two constraints of (12.5), the column of the constraint matrix corresponding to arc (i, j) has a +1 coefficient in the i th constraint, a -1 coefficient in the j th constraint, and zeros in every other component. This implies that the rank of the constraint matrix is not more than $n - 1$ since adding all rows together generates a vector of zeros.

To justify our thinking that linearly independent columns correspond to

arcs in a spanning tree, let's consider the minimum cost flow problem given in [Figure 12.17](#), and consider the spanning tree $T = \{(1, 4), (3, 2), (4, 3), (5, 2)\}$, given in [Figure 12.18](#). The submatrix B_T of the constraint matrix corresponding to the

FIGURE 12.18 Spanning tree example for MCNF problem.



arcs of T is

$$B_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

If we rearrange its rows (which does not affect the rank of B_T) in the order of the nodes 1, 4, 3, 2, 5, and the columns in the order (1, 4), (4, 3), (3, 2), (5, 2), we get the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

It is easy to see that this matrix has rank $n - 1 = 4$. Now, consider the set of arcs (1, 2), (1, 4), (4, 5) and (5, 2) generating a cycle in G (if we remove the arc directions). They generate the submatrix

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}.$$

Note that this submatrix has a rank less than 4 since

$$\begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

We can generalize this approach.

Theorem 12.2 Let A be the constraint matrix of the minimum cost network flow problem (<12.5). Then the rank of A is $n - 1$, and every collection of $n - 1$ linearly independent columns of A corresponds to a spanning tree of G .

Proof Let's first determine the rank of A . Suppose T is the set of arcs belonging to some spanning tree of G and let B_T be the $n \times (n - 1)$ submatrix of A whose columns correspond to the arcs of T . Because T is a tree, there exists at least one node i incident to exactly one arc of T , which implies that the row of B_T corresponding to i contains exactly one nonzero element. Reorder the rows and columns of B_T so that the first row now belongs to node i and the first column to the edge incident to i in T . This results in the matrix

$$B_T = \begin{bmatrix} \pm 1 & \mathbf{0} \\ \mathbf{p} & B_{T_1} \end{bmatrix},$$

where B_{T_1} is the submatrix of A whose rows correspond to all nodes $V - \{i\}$ and whose arcs correspond to a spanning tree over these nodes. Repeating this process leads to a reordering of the rows and columns of B_T where the (k, k) -elements of B_T are all ± 1 and every element (i, j) where $i < j$ has value 0. It is easy to see that this matrix has rank $n - 1$, and so A has rank $n - 1$.

To show that every collection of $n - 1$ linearly independent columns of A corresponds to a spanning tree of G , suppose there exists a set of $n - 1$ linearly independent columns of A that contains a cycle, and let W be those arcs in the cycle. Note that the column \mathbf{a}_{ij} of A corresponding to the arc (i, j) can be written as

$$\mathbf{a}_{ij} = \mathbf{e}_i - \mathbf{e}_j,$$

where \mathbf{e}_k is the unit vector with a 1 in the k th coordinate. Let (i, j) be an arc on this cycle, and suppose we orient the cycle in the direction from i to j .

Define, for each $(k, l) \in W$

$$\delta_{kl} = \begin{cases} 1, & \text{if arc } (k, l) \text{ is in the direction of orientation of the cycle,} \\ -1, & \text{if arc } (k, l) \text{ is in opposite direction of orientation of the cycle,} \end{cases}$$

then we have

$$\sum_{(k,l) \in W} \delta_{kl} a_{kl} = \sum_{(k,l) \in W} \delta_{kl} (e_k - e_l) = \mathbf{0}.$$

Theorem 12.2 shows that similar to the transportation problem, every extended basic solution to (12.5) corresponds to a spanning tree T of G , along with arcs U whose flow values are at their upper bound, and arcs L whose values are at their lower bounds. Similar to the transportation simplex, this implies the following:

1. When we find an entering variable, this corresponds to adding an arc to our spanning tree, which generates a unique cycle.
2. A leaving variable corresponds to the removal of an arc on this unique cycle.

■ EXAMPLE 12.8

Consider the minimum cost flow problem given in [Figure 12.17](#), and suppose we have the initial flow $x_{12} = x_{13} = x_{14} = x_{43} = 5$, $x_{32} = 10$. This flow corresponds to an extended basic feasible solution with $U = \{(1, 2), (1, 3)\}$, $L = \{(4, 5), (5, 3)\}$, and $T = \{(1, 4), (4, 3), (3, 2), (5, 2)\}$; note that this solution is degenerate since we have the basic variable $x_{52} = 0$. This solution is given in [Figure 12.19](#).

Once we have an initial basic feasible solution, we need to calculate the reduced costs for the nonbasic variables in L and U to determine if we are at the optimal solution. If we assign dual variables y_i to the constraints (12.5b) for each $i \in V$, we get the dual constraints

$$y_i - y_j \leq c_{ij},$$

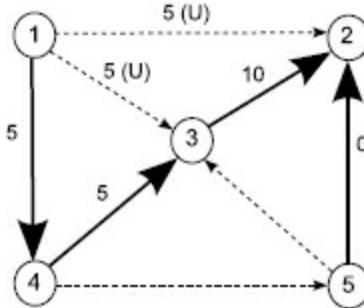
for each $(i, j) \in A$. Thus, our reduced costs are

$$\bar{c}_{ij} = c_{ij} - y_i + y_j.$$

As we did in the transportation simplex method, we can calculate the values

of the dual variables y_i by setting the reduced cost of all basic variables $(i, j) \in T$ equal to 0; since there are n variables and $n - 1$ equalities, we can arbitrarily fix one of the variables to 0 and uniquely solve for the remainder.

FIGURE 12.19 Initial extended basic feasible solution for Example 12.8.



■ EXAMPLE 12.9

Continuing with Example 12.8, using the basic variables we first derive the dual variable values by solving the system

$$\begin{array}{rcl} 2 - y_1 & + y_4 & = 0 \\ 5 & + y_3 - y_4 & = 0 \\ 2 & + y_2 - y_3 & = 0 \\ 4 & + y_2 & - y_5 = 0. \end{array}$$

If we initially fix $y_1 = 0$ and solve for the remaining variables, we get $y_2 = -9$, $y_3 = -7$, $y_4 = -2$, and $y_5 = -5$. Using these dual values, we get the reduced costs

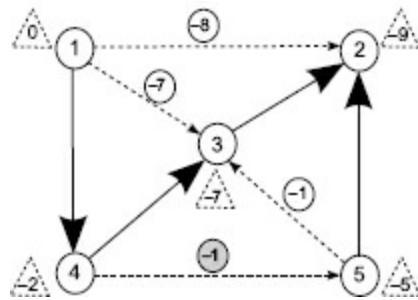
$$\begin{aligned} \bar{c}_{12} &= 1 - 0 + (-9) = -8 \\ \bar{c}_{13} &= 0 - 0 + (-7) = -7 \\ \bar{c}_{45} &= 2 - (-2) + (-5) = -1 \\ \bar{c}_{53} &= 1 - (-5) + (-7) = -1. \end{aligned}$$

These calculations are shown in [Figure 12.20](#), where the numbers in the triangles are the dual costs for each node and the circled numbers along the arcs are the reduced costs.

Since $(1, 2), (1, 3) \in U$, their reduced costs satisfy the optimality condition, but we see that either x_{45} or x_{53} can be entering variables, since they are at their lower bounds with negative reduced costs. If we choose x_{45} to be our entering variable (denoted in [Figure 12.20](#) by the shaded reduced

cost), we get the cycle arcs $(4, 5)$, $(5, 2)$, $(3, 2)$, and $(4, 3)$.

FIGURE 12.20 Reduced cost calculations.



Once we have identified an entering variable, a cycle in our network has been created; our leaving variable corresponds to the arc that is removed from this cycle to create a new spanning tree. As with the transportation simplex method, we want to increase the value of the entering variable to θ . If we orient the cycle in the direction of our entering variable arc, every arc on the cycle in the direction of the cycle orientation also has its value increase by θ , and the arcs with opposite direction to the cycle orientation has its value decrease by θ . Hence, we need to identify the value of θ that ensures all variable bounds remain satisfied. Our leaving variable corresponds to one of the arcs that are at one of their bounds due to the choice of θ .

■ EXAMPLE 12.10

If we continue from Example 12.9, we see that our cycle arcs are $(4, 5)$, $(5, 2)$, $(3, 2)$, and $(4, 3)$. Since $(4, 5)$ corresponds to our entering variable, and we have $u_{45} = 10$, we know that $\theta \leq 10$. Arc $(5, 2)$ is in the same direction as the cycle orientation, so its value increases from 0 to $0 + \theta$; due to its capacity $u_{52} = 10$, this implies (again) that $\theta \leq 10$. Arcs $(3, 2)$ and $(4, 3)$ have direction opposite the orientation of the cycle, so their values decrease by θ to $x_{32} = 10 - \theta$ and $x_{43} = 5 - \theta$; since these values must remain nonnegative, this implies that $\theta \leq 10$ and $\theta \leq 5$. The largest value of θ is 5, corresponding to arc $(4, 3)$, and implying that variable x_{43} becomes nonbasic and $(4, 3)$ moves to L . Our new basic feasible solution has values $x_{12} = x_{13} = x_{14} = 5$, $x_{45} = 5$, $x_{32} = 5$, and $x_{52} = 5$, with basis structure

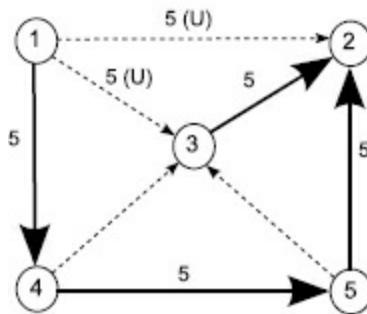
$$T = \{(1, 4), (4, 5), (3, 2), (5, 2)\}$$

$$\mathcal{L} = \{(4, 3), (5, 3)\}$$

$$\mathcal{U} = \{(1, 2), (1, 3)\}.$$

The new solution is shown in [Figure 12.21](#).

FIGURE 12.21 Basic feasible solution found in Example 12.10.



Now that we've seen the various elements of the bounded simplex method specialized for the minimum cost network flow problem, we can formally state the **Network Simplex Method** which is given in Algorithm 12.8.

Algorithm 12.8 Network Simplex Method

Step 1: Obtain an initial extended basic feasible solution (T, L, U) , where T is a spanning tree of G .

Step 2: Use the fact that $c_{ij} = y_i - y_j$ for all basic variables x_{ij} , $(i, j) \in T$, and that one variable, say y_1 , can be fixed to 0, to find all dual variables for the current basic feasible solution.

Step 3: Compute the reduced costs $\bar{c}_{ij} = c_{ij} - y_i + y_j$ for all $(i, j) \in L \cup U$.

Step 4: If $\bar{c}_{ij} \geq 0$ for all $(i, j) \in L$ and $\bar{c}_{ij} \leq 0$ for all $(i, j) \in U$, STOP. We are at an optimal solution. Otherwise, choose an entering variable among those nonbasic variables violating these conditions.

Step 5: Using the unique cycle formed by adding the entering variable, identify the leaving variable and update the basic feasible solution (T, L, U) .

Step 6: Using the new basic feasible solution, return to Step 2.

■ EXAMPLE 12.11

Let's finish the network simplex method from where we ended in Example 12.10. Using the basic variables $T = \{(1, 4), (4, 5), (3, 2), (5, 2)\}$, we first derive the dual variable values by solving the system

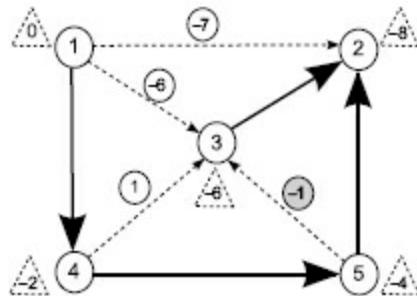
$$\begin{aligned}
 2 - y_1 & + y_4 = 0 \\
 2 & - y_4 + y_5 = 0 \\
 2 & + y_2 - y_3 = 0 \\
 4 & + y_2 - y_5 = 0.
 \end{aligned}$$

If we initially fix $y_1 = 0$ and solve for the remaining variables, we get $y_2 = -8$, $y_3 = -6$, $y_4 = -2$, and $y_5 = -4$. Using these dual values, we get the reduced costs

$$\begin{aligned}
 \bar{c}_{12} &= 1 - 0 + (-8) = -7 \\
 \bar{c}_{13} &= 0 - 0 + (-6) = -6 \\
 \bar{c}_{43} &= 5 - (-2) + (-6) = 1 \\
 \bar{c}_{53} &= 1 - (-4) + (-6) = -1.
 \end{aligned}$$

These calculations are shown in [Figure 12.22](#), where again the numbers in the triangles are the dual costs for each node and the circled numbers along the arcs are the reduced costs.

FIGURE 12.22 Reduced cost calculations (2nd iteration).



Our entering variable is x_{53} , since it is the only variable violating the optimality condition. Adding $(5, 3)$ to our spanning tree yields the cycle arcs $(5, 3)$, $(3, 2)$, and $(5, 2)$. Since $(5, 3)$ corresponds to our entering variable, and we have $u_{53} = 10$, we know that $\theta \leq 10$. Arc $(3, 2)$ is in the same direction as the cycle orientation, so its value increases from 5 to $5 + \theta$; due to its capacity $u_{52} = 10$, this implies (again) that $\theta \leq 5$. Arc $(5, 2)$ has direction opposite the orientation of the cycle, so its value decreases by θ to $x_{52} = 5 - \theta \geq 0$, implying that $\theta \leq 5$. Hence, we have the largest value of θ at 5, corresponding to arcs $(3, 2)$ and $(5, 2)$; we shall choose x_{52} as our leaving variable and assign $(5, 2)$ to L . Our new basic feasible solution has values $x_{12} = x_{13} = x_{14} = 5$, $x_{45} = 5$, $x_{53} = 5$, and $x_{32} = 10$, with basis structure

$$T = \{(1, 4), (4, 5), (3, 2), (5, 3)\}$$

$$\mathcal{L} = \{(4, 3), (5, 2)\}$$

$$\mathcal{U} = \{(1, 2), (1, 3)\}.$$

This new solution is shown in [Figure 12.23](#).

FIGURE 12.23 Basic feasible solution found in Example 12.10.

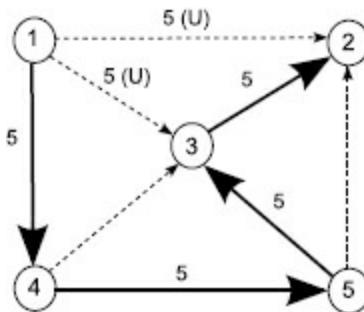
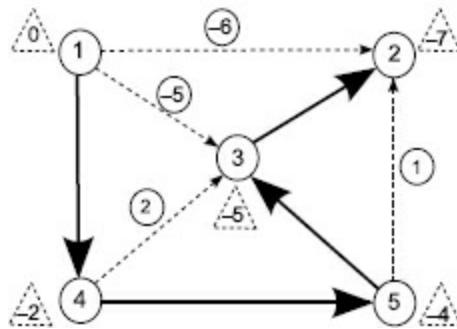


FIGURE 12.24 Optimal solution found in Example 12.10.



From the basic variables T , we can compute the new dual variable values by solving

$$\begin{aligned} 2 - y_1 &+ y_4 &= 0 \\ 2 &- y_4 + y_5 &= 0 \\ 2 &+ y_2 - y_3 &= 0 \\ 1 &+ y_3 &- y_5 = 0. \end{aligned}$$

which, after setting $y_1 = 0$, generates the solution $y_1 = 0, y_2 = -7, y_3 = -5, y_4 = -2$, and $y_5 = -4$. The reduced costs for the nonbasic variable arcs are given in [Figure 12.24](#). Since all reduced costs satisfy the optimality conditions, this is our optimal solution.

The network simplex method is a computationally efficient algorithm for solving minimum cost network flow problems. However, it has one important

drawback that needs to be addressed. Some computational studies have shown that a large percentage of iterations may be degenerate, which can lead to cycling. Therefore, special care is needed in order to guarantee the algorithm will end after a finite number of iterations. If this is done, the network simplex method becomes one of the faster algorithms in practice for solving this problem.

Summary

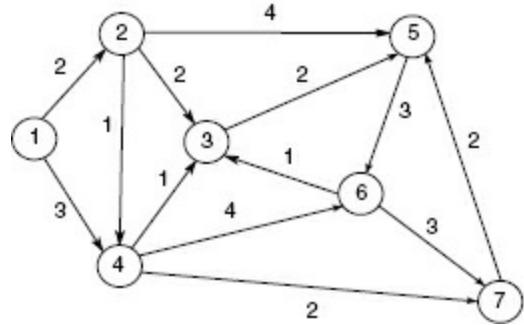
In this chapter, we derived exact algorithms for three different network optimization problems—shortest path problem, maximum flow problem, and minimum cost network flow problem. In each case, we first established an optimality condition for the problem by exploring an example problem, and then used this condition to obtain an algorithm. We then exploited the mathematical structure of the problem to improve these algorithms. This approach is often used to design algorithms for combinatorial problems. In fact, we could have continued our studies of these problems to obtain more diverse algorithms for the problems. For more examples of algorithms for these and other network flow problems, the book by Ahuja, Magnanti, and Orlin [2] provides more details not only on these algorithms but also on their implementations.

EXERCISES

12.1 Solve the shortest path problem (using $s = 1$) given in [Figure 12.25](#) using

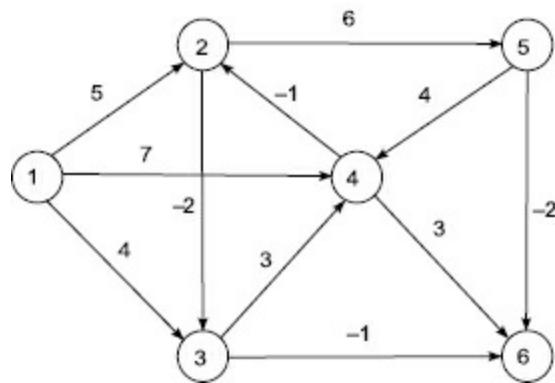
- (a)** FIFO label-correcting algorithm.
- (b)** Dijkstra's algorithm.

[**FIGURE 12.25**](#) Shortest path problem for Exercise 12.1.



12.2 Solve the shortest path problem (using $s = 1$) given in [Figure 12.26](#) using FIFO label-correcting algorithm.

FIGURE 12.26 Shortest path problem for Exercise 12.2.



12.3 Solve the shortest path problem (using $s = 1$) given in [Figure 12.26](#) using Dijkstra's method. Why are the resulting lengths different from those obtained in Exercise 12.2?

12.4 Give an example of a shortest path problem where some of the arc lengths are negative but where Dijkstra's method generates the optimal path lengths.

12.5 Consider the network given in [Figure 12.27](#). Show that if we use the modified label-correcting algorithm (with $s = 6$) and always select the node i from $List$ with minimum index, then node 1 gets labeled 16 times.

FIGURE 12.27 Worst-case example for modified label-correcting algorithm.

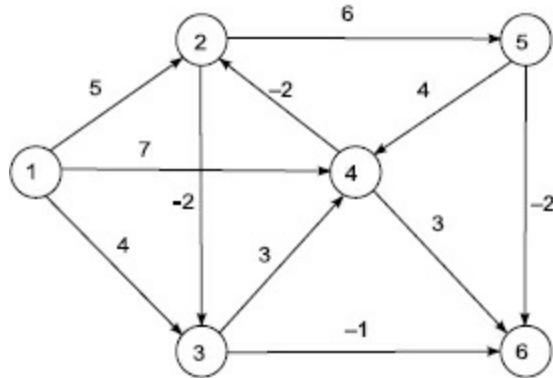
```

 $d(s) \leftarrow 0$  and  $pred(s) \leftarrow -1$ .
 $d(j) \leftarrow M$  and  $pred(j) = -1$  for each  $j \in V - \{s\}$ 
List =  $\{s\}$ .
repeat
    Remove  $i$  from List
    for all  $(i, j) \in A(i)$  do
        if  $d(j) > d(i) + c_{ij}$  then
             $d(j) = d(i) + c_{ij}$ .
             $pred(j) = i$ .
            if  $j \notin List$  then
                List  $\leftarrow List \cup \{j\}$ 
            end if
        end if
    end for
until List =  $\emptyset$ 

```

12.6 How can we use the FIFO label-correcting algorithm to find negative cycles if they exist in the network? Illustrate your method on [Figure 12.28](#).

FIGURE 12.28 Shortest path problem for Exercise 12.6.



12.7 For the network given in [Figure 12.29](#), find the maximum flow from node 1 to node 7 using the labeling algorithm and identify a minimum cut.

12.8 For the network given in [Figure 12.30](#), find the maximum flow from node s to node t using the labeling algorithm and identify a minimum cut.

FIGURE 12.29 Maximum flow problem for Exercise 12.7.

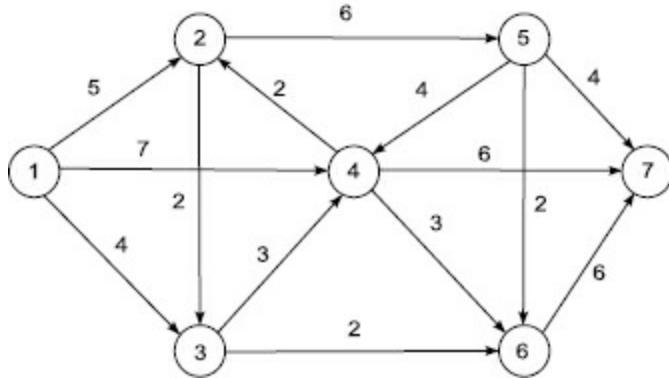
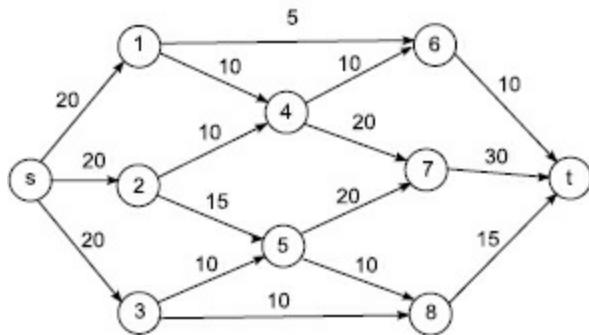


FIGURE 12.30 Maximum flow problem for Exercise 12.8.



12.9 Suppose in Exercise 12.8, we want to limit the amount of flow through node 5 to at most 15 units. How can we model this? Illustrate and resolve the problem to obtain a new maximum flow.

12.10 For the network given in [Figure 12.31](#), find the maximum flow from node s to node t using the labeling algorithm and identify a minimum cut. Why is the maximum flow value not equal to 5?

12.11 Suppose we are given a maximum flow problem with multiple sources s_1, s_2, \dots, s_p and multiple sinks t_1, t_2, \dots, t_q . How can we formulate this problem into one with only one source s and one sink t ?

12.12 Solve the minimum cost network flow problem given in [Figure 12.32](#) using the cycle- canceling algorithm, starting with the initial solution $x_{12} = 5, x_{13} = 10, x_{24} = 10, x_{34} = 30, x_{35} = 15, x_{45} = 10, x_{46} = 10, x_{52} = 5$, and $x_{56} = 20$.

12.13 Solve the minimum cost network flow problem given in [Figure 12.32](#), using the network simplex algorithm. Start with the initial solution $x_{12} = 5, x_{13} = 10, x_{24} = 10, x_{34} = 30, x_{35} = 15, x_{45} = 10, x_{46} = 10, x_{52} = 5$,

and $x_{56} = 20$.

FIGURE 12.31 Maximum flow problem for Exercise 12.10.

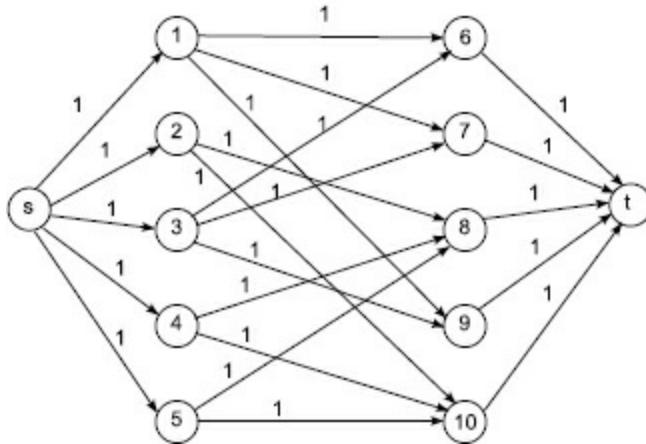
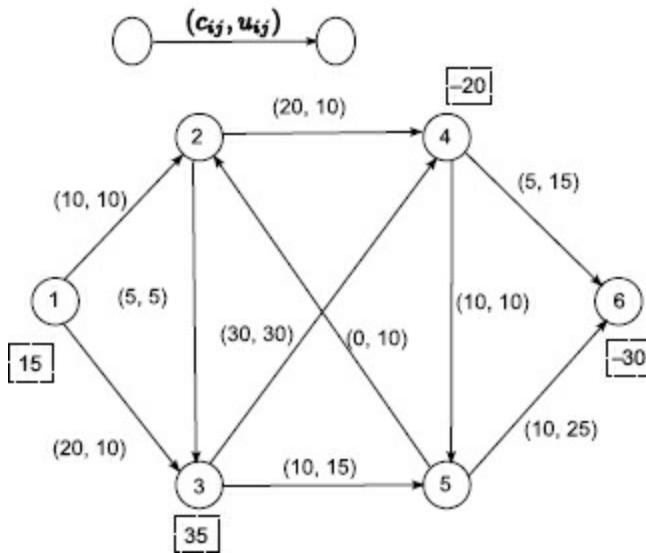
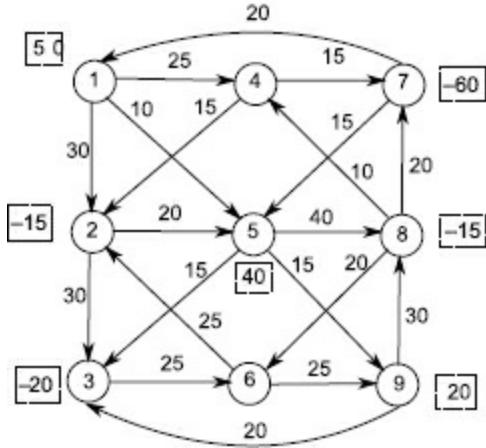


FIGURE 12.32 Minimum cost flow problem for Exercises 12.12 and 12.13.



12.14 Consider the minimum cost network flow problem given in [Figure 12.33](#), which provides the cost c_{ij} of sending each unit of flow on arc (i, j) ; there is no upper bound on the amount of flow that can be sent on each arc, and the dashed boxes near a node indicate its supply or demand. Use the network simplex algorithm to solve this problem.

FIGURE 12.33 Minimum cost flow problem for Exercises 12.14.



12.15 In Example 2.12, we introduced the minimum cost network flow model using the PedalMetal Motors problem. Solve this model using the cycle-canceling algorithm, using the initial feasible solution $x_{Det, Den} = x_{Atl, Den} = x_{Atl, Cin} = 50$, $x_{Det, Cin} = 60$, $x_{Den, LA} = x_{Den, Chi} = x_{Cin, LA} = 40$, and $x_{Den, Phi} = x_{Cin, Chi} = x_{Cin, Phi} = 30$.

12.16 Suppose $d(i)$ is the distance of some path from node s to node i and let

$$c_{ij}^d = c_{ij} + d(i) - d(j).$$

(a) Show that for any directed cycle W in the graph G ,

$$\sum_{(i,j) \in W} c_{ij}^d = \sum_{(i,j) \in W} c_{ij}.$$

(b) Show that for any directed path P from node k to node l ,

$$\sum_{(i,j) \in P} c_{ij}^d = \sum_{(i,j) \in P} c_{ij} + d(k) - d(l).$$

12.17 When studying the maximum flow problem, we assumed that there does not exist a path from s to t where every arc (i, j) on the path had infinite capacity. Does this mean that there can only be arcs with finite capacity in our problem? Either prove this statement or provide an example where at least one arc has infinite capacity but the maximum amount of flow sent from s to t is finite.

12.18 Find the dual of the linear programming formulation (12.4) of the maximum flow problem and show that its optimal solution corresponds to a minimum cut.

12.19 Given a minimum cost network flow problem (12.5), set up a maximum flow problem whose solution helps determine whether 12.5) has a feasible solution. Illustrate this formulation on the network given in [Figure 12.32](#).

CHAPTER 13

INTRODUCTION TO INTEGER PROGRAMMING

Until now we have worked primarily with linear optimization problems where the variables are allowed to have fractional values. These *continuous optimization problems*, especially linear programs, have been studied extensively, and algorithms that attempt to find local optima have been known for many years. In some cases, we know that global optima are easy to find. For *discrete optimization problems*, where certain variables are constrained to only have integer values, these techniques do not help us. While it may seem counterintuitive, allowing more values in the continuous problems makes these problems easier to solve than discrete problems that have far fewer feasible solutions. In fact, most discrete problems cannot be solved like linear programs because there are no general optimality conditions for them. One implication of this is that discrete problems can take enormous amounts of time and effort to solve, even for a small number of variables. Thus, we must find new ways to solve discrete optimization problems.

In the next three chapters, we examine various solution approaches for discrete linear optimization problems and illustrate how important continuous linear programs are to solving these problems. In this chapter, after recalling the definition of an integer program, we discuss why these problems are difficult to solve in general and how important it is to properly describe the feasible region. We then reintroduce the idea of a relaxed problem and its role in solution methods. Finally, we discuss some known cases where integer programs are actually easy to solve. Once we have completed this chapter, we are ready to discuss actual methods to solve integer programs.

13.1 BASIC DEFINITIONS AND FORMULATIONS

We have been introduced to integer programs previously, especially in Chapters 3 and 4 where we modeled various problems as integer programs. Before we begin a more formal discussion of solution techniques, it is probably best to recall the definitions of these problems.

Mixed-Integer Program Given a linear program

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & A\mathbf{x} \leq \mathbf{b} \\ (13.1) \quad & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

if at least one of the variables x_i must be an integer, problem (13.1) is said to be an **mixed-integer program**. If S denotes the set of discrete variables, then (13.1) would be written as

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ & x_i \text{ integer, } i \in S. \end{aligned}$$

Integer Program If all the variables must have integer values in (13.1), then it is referred to as a **integer program**. It would then be written as

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & A\mathbf{x} \leq \mathbf{b} \\ (13.2) \quad & \mathbf{x} \geq \mathbf{0}, \text{ integer.} \end{aligned}$$

Binary or 0–1 Program An integer program where all the variables are bounded by 0 and 1 is known as a **binary or 0–1 program**.

Throughout the next two chapter, we will assume that **every component of A , c , and b has integer value**. This is not a large restriction, since computers represent every irrational number by an approximate rational

number, and a set of rational numbers can be made simultaneously integer values by multiplying each by a large enough integer.

An important consideration we need to make with regard to integer programs is how we formulate them. This was not much of an issue with linear programs because we knew that an optimal solution existed at an extreme point, and we could identify them by looking at the active constraints. However, for integer programs, it is not as easy.

■ EXAMPLE 13.1

Consider the linear program

$$\begin{aligned} \max \quad & 8x + 7y \\ \text{s.t.} \quad & -18x + 38y \leq 133 \\ & 13x + 11y \leq 125 \\ & 10x - 8y \leq 55 \\ (13.3) \quad & x, y \geq 0, \end{aligned}$$

whose feasible region is given in [Figure 13.1](#). Note that all the extreme points have fractional values, while most of the integer points occur within the interior of the feasible region. This implies that if we were to solve (13.3) by the simplex method, we would not have an integer solution.

FIGURE 13.1 Feasible region for integer program (13.3).

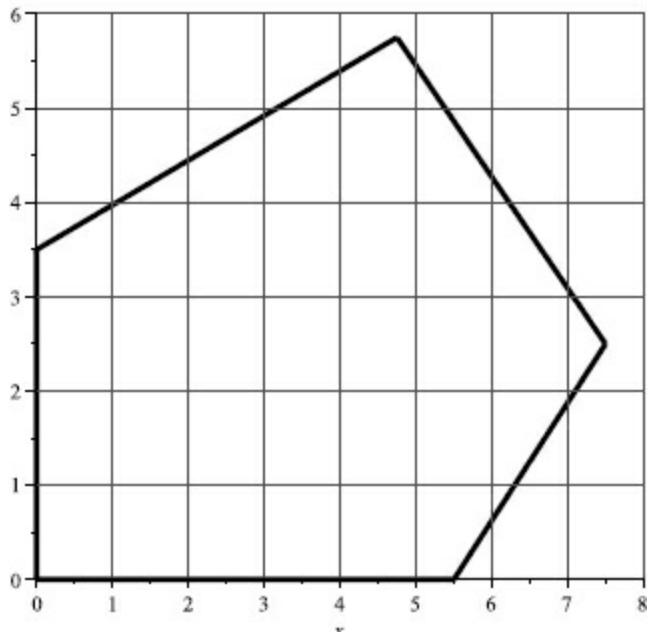
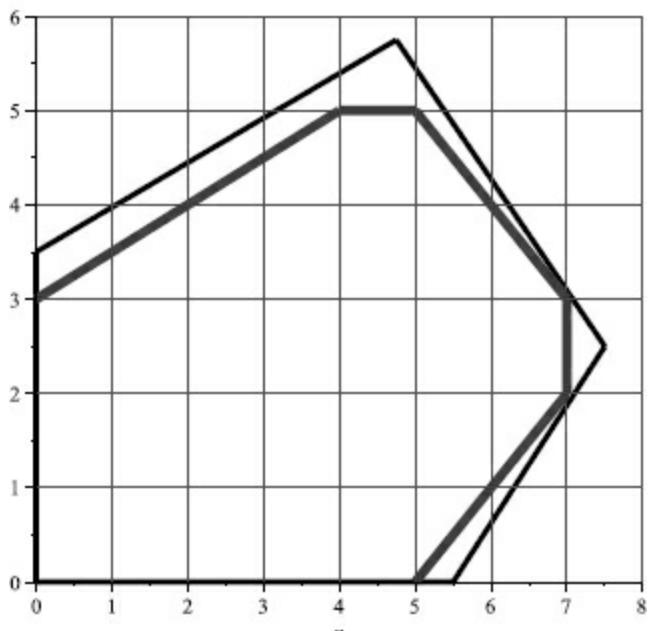


FIGURE 13.2 Feasible region for integer program (13.4).



However, consider the linear program

$$\begin{aligned}
\max \quad & 8x + 7y \\
\text{s.t.} \quad & -x + 2y \leq 6 \\
& x + y \leq 10 \\
& x - y \leq 5 \\
& x \leq 7 \\
& y \leq 5, \\
(13.4) \quad & x, y \geq 0
\end{aligned}$$

whose feasible region is given in [Figure 13.2](#). Note that the “interior” feasible region contains all the integer solutions to (13.3) and that all its extreme points are integer valued. Thus, if we were to solve (13.4) as a linear program (without the integrality constraints), we would have an integer solution that would be an optimal solution to the integer program (13.3).

This example illustrates that (1) integer programs are often hard because we cannot “see” the feasible integer solutions as we can the continuous solutions (via extreme points) and (2) if we have integer-valued extreme points, we can solve integer programs by solving a linear program. This bears repeating.

Given an integer program (13.2), if we remove the integrality constraints and consider only its linear program relaxation (13.1), and the optimal solution to (13.1) has only integer values, then this solution is also a solution to (13.2).

■ EXAMPLE 13.2

Consider the integer program

$$\begin{aligned}
\max \quad & z = 13x + 5y \\
\text{s.t.} \quad & 4x + y \leq 24 \\
& x + 3y \leq 24 \\
& 3x + 2y \leq 23 \\
& x, y \geq 0, \text{ integer}.
\end{aligned}$$

We saw in Chapter 1 that, if we remove the integrality constraint, the feasible region to the resulting linear program has integer extreme points. Hence, solving the linear program

$$\max \quad z = 13x + 5y$$

s.t.

$$4x + y \leq 24$$

$$x + 3y \leq 24$$

$$3x + 2y \leq 23$$

$$x, y \geq 0$$

yields an optimal solution to the integer program.

Ideally, we would like to solve integer programs as linear programs since we know how to do this. For this approach to work, we must write our integer program in such a way that all its extreme points (when integrality is relaxed) are integer valued. To formalize this idea more, we need the following definitions. Recall from Chapter 7 that a *polyhedron* is a subset of \mathbb{R}^n described by a finite set of linear constraints.

Formulation A polyhedron $P \subset \mathbb{R}^n$ is a *formulation* for a set $X \subset \mathbb{Z}^n$ if

$X = P \cap \mathbb{Z}^n$. In other words, given a set of integer-valued solutions X , if we can find a polyhedron P such that $X \subset P$ and there are no integer-valued solutions in P that are not in X , then P is a formulation of X .

■ EXAMPLE 13.3

Consider the set of solutions

$$X = \left\{ (0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (0, 1, 1, 0), (0, 1, 0, 1), (0, 0, 1, 1) \right\}.$$

One formulation of this set is

$$P = \left\{ \mathbf{x} \in \mathbb{R}^4 : \begin{array}{l} 83x_1 + 57x_2 + 40x_3 + 19x_4 \leq 100 \\ 0 \leq x_i \leq 1, \quad i = 1, 2, 3, 4 \end{array} \right\}.$$

It should be clear that there are no integer-valued solutions in P that are not in X and that every solution in X is also in P .

Often, multiple formulations of a given set exist, so it is useful to determine which one to use.

Better Formulation Given a set X and two formulations P_1, P_2 for X ,

P_1 is a *better formulation* than P_2 if $P_1 \subset P_2$.

Ideal Formulation Given a set X , a formulation P for X is *ideal* if each extreme point of P is an element of X .

Consider the integer program

$$\max\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in X\}.$$

If P is an ideal formulation of X , then solving the linear program

$$\max\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in P\}$$

yields an optimal solution to the original integer program.

When writing integer programs, our goal is always to construct an ideal formulation of the problem. However, this is often very difficult, if not impossible. Hence, we often try to find better formulations of our problem.

■ EXAMPLE 13.4

Consider the set of points

$$X = \left\{ (0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (0, 1, 1, 0), (0, 1, 0, 1), (0, 0, 1, 1) \right\}.$$

In Example 13.3 we gave the formulation

$$P_1 = \left\{ \mathbf{x} \in \mathbb{R}^4 : \begin{array}{l} 83x_1 + 57x_2 + 40x_3 + 19x_4 \leq 100 \\ 0 \leq x_i \leq 1, \quad i = 1, 2, 3, 4 \end{array} \right\}$$

of X . A better formulation of this set is

$$P_2 = \left\{ \mathbf{x} \in \mathbb{R}^4 : \begin{array}{l} 6x_1 + 4x_2 + 3x_3 + 2x_4 \leq 7 \\ 0 \leq x_i \leq 1, \quad i = 1, 2, 3, 4 \end{array} \right\}$$

since $P_2 \subset P_1$. It turns out that an ideal formulation of X is

$$P_3 = \left\{ \mathbf{x} \in \mathbb{R}^4 : \begin{array}{l} x_1 + x_2 \leq 1 \\ x_1 + x_3 \leq 1 \\ x_1 + x_4 \leq 1 \\ 2x_1 + x_2 + x_3 + x_4 \leq 2 \\ 0 \leq x_i \leq 1, \end{array} \right\}.$$

How can we find (at least theoretically) an ideal formulation? One way is to go back to the idea of a convex hull of a set of solutions that was first introduced in Chapter 7. If we let $X = \{ \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k \}$ be a set of solutions in \mathbb{R}^n , then the set

$$S = \text{conv}(X) = \left\{ \mathbf{x} : \mathbf{x} = \sum_{i=1}^k \alpha_i \mathbf{v}_i, \sum_{i=1}^k \alpha_i = 1, \alpha_i \geq 0 \text{ for all } i \right\}$$

is the **convex hull** of X . If we can find the convex hull of a set of integer solutions X , we are guaranteed to have (a) a polytope, and hence a convex set, and (b) each extreme point is a solution in X .

If X is a finite set of integer-valued solutions in \mathbb{R}^n , then the convex hull of X is always an ideal formulation of X . Hence, theoretically we can solve any integer program using linear programming techniques.

■ EXAMPLE 13.5

If we consider again the set of solutions X given in Example 13.1, the convex hull of X can be defined as

$$\text{conv}(X) = \left\{ (x, y) : \begin{array}{l} -x + 2y \leq 6 \\ x + y \leq 10 \\ x - y \leq 5 \\ x \leq 7 \\ y \leq 5 \\ x, y \geq 0 \end{array} \right\}.$$

Hence, solving an integer program over X is equivalent to solving a linear program whose feasible region is $\text{conv}(X)$.

Unfortunately, this is primarily a theoretical result since finding the convex hull of a set of solutions is very difficult. In fact, it often happens that the number of linear constraints needed to define the convex hull is very large (i.e., exponential), and hence we cannot even find them all. Because of this, we often do not look for ideal formulations. This does not mean that we don't look for better formulations; in fact, often multiple formulations of the same integer program are given and compared to see which ones are better, as we

saw in Example 13.4.

■ EXAMPLE 13.6

In a fixed-charge problem, given a collection of variables $x_k \in \{0, 1\}, k = 1, \dots, n$, we want variable $y = 1$ if at least one of the x_k 's is equal to 1. Two formulations of this are

$$P_1 = \left\{ (x_1, x_2, \dots, x_n, y) \in \{0, 1\}^{n+1} : \sum_{k=1}^n x_k \leq ny \right\}$$
$$P_2 = \left\{ (x_1, x_2, \dots, x_n, y) \in \{0, 1\}^{n+1} : x_k \leq y, k \in \{1, \dots, n\} \right\}.$$

We can easily show that $P_2 \subset P_1$, and hence P_2 is a better formulation than P_1 (see Exercise 13.5).

13.2 RELAXATIONS AND BOUNDS

If we do not have an ideal formulation of the feasible region to your integer program, how do we know if our current feasible integer solution is optimal? For linear programs, optimality conditions provided the answer to this question. Unfortunately there is no optimality condition for an integer program, so it is not as simple.

One approach we can take is to find an upper bound z_{UB} to our problem and compare it to the feasible solution's value $z_{current}$; we first saw this approach in Chapter 5 when we explored heuristic methods. If $z_{UB} - z_{current} < \epsilon$, for ϵ “small enough,” then the current solution is optimal. Since all objective coefficients are integers, we can set $\epsilon = 1$ for this test. But how do we find such an upper bound? Typically, we modify our current problem into one that is (a) easily solved and (b) contains all the current feasible solutions to our integer program.

Relaxation Given any optimization problem P , a *relaxation* P_R of this

problem is one where every feasible solution to P is also feasible to P_R and every feasible solution \mathbf{x} in P has value no worse in P_R .

Let P denote a maximization optimization problem and P_R a relaxation of it. Let $z(P)$ denote the optimal value of P , and let $z(P_R)$ denote the optimal value of P_R . Then

$$z(P) \leq z(P_R).$$

Relaxation Gap The difference

$$z(P_R) - z(P)$$

is called the *relaxation gap*.

One example of a relaxation of an integer program (13.2) is a **linear programming relaxation (LP-relaxation)**, where the integrality constraints are removed.

■ EXAMPLE 13.7

Consider the integer program

$$\begin{aligned} z_{IP}^* = \max \quad & 8x + 7y \\ \text{s.t.} \quad & -18x + 38y \leq 133 \\ & 13x + 11y \leq 125 \\ & 10x - 8y \leq 55 \\ & x, y \geq 0, \text{ integer.} \end{aligned}$$

Its linear programming relaxation

$$\begin{aligned} z_{LP}^* = \max \quad & 8x + 7y \\ \text{s.t.} \quad & -18x + 38y \leq 133 \\ & 13x + 11y \leq 125 \\ & 10x - 8y \leq 55 \\ & x, y \geq 0 \end{aligned}$$

has optimal value $z_{LP}^* = 78.25$. Since the objective function coefficients are integers, the optimal value of the integer program cannot be more than 78. In fact, solving the integer program yields an optimal value of 77. If the objective function was $8x + 6y$, the linear programming relaxation value is 74.9533 while the optimal integer value is 74.

Another common relaxation technique is **Lagrangian relaxation**, where some (or all) constraints are “dualized” and brought into the objective function. We saw this duality approach in Chapter 9. For example, if our integer program is of the form

$$\begin{aligned}
 \max \quad & \mathbf{c}^T \mathbf{x} \\
 \text{s.t.} \quad & \\
 & A_1 \mathbf{x} \leq \mathbf{b}_1 \\
 & A_2 \mathbf{x} = \mathbf{b}_2 \\
 (13.5) \quad & \mathbf{x} \geq \mathbf{0}, \text{ integer,}
 \end{aligned}$$

one Lagrangian relaxation of (13.5) could be

$$\begin{aligned}
 z_{LD}^* = \min_{\lambda \geq \mathbf{0}} \max & \quad \mathbf{c}^T \mathbf{x} + \lambda_1^T (\mathbf{b}_1 - A_1 \mathbf{x}) \\
 \text{s.t.} \quad & \\
 & A_2 \mathbf{x} = \mathbf{b}_2 \\
 (13.6) \quad & \mathbf{x} \geq \mathbf{0}, \text{ integer.}
 \end{aligned}$$

In this case, we would leave the set of constraints $A_2 \mathbf{x} = \mathbf{b}_2$ if they can be easily solved over integer variables, such as for transportation constraints or any network-based constraints. Note that if \mathbf{x} is feasible to (13.5), it is also feasible to (13.6) and the optimal value of (13.6) must be at least as large as that of (13.5). Lagrangian relaxations are typically not used for general problems, but only for specifically structured problems. This is in contrast to LP-relaxations, which can be (and are) used for all problems.

■ EXAMPLE 13.8

Consider the 0–1 knapsack problem

$$\max \quad 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5$$

s.t.

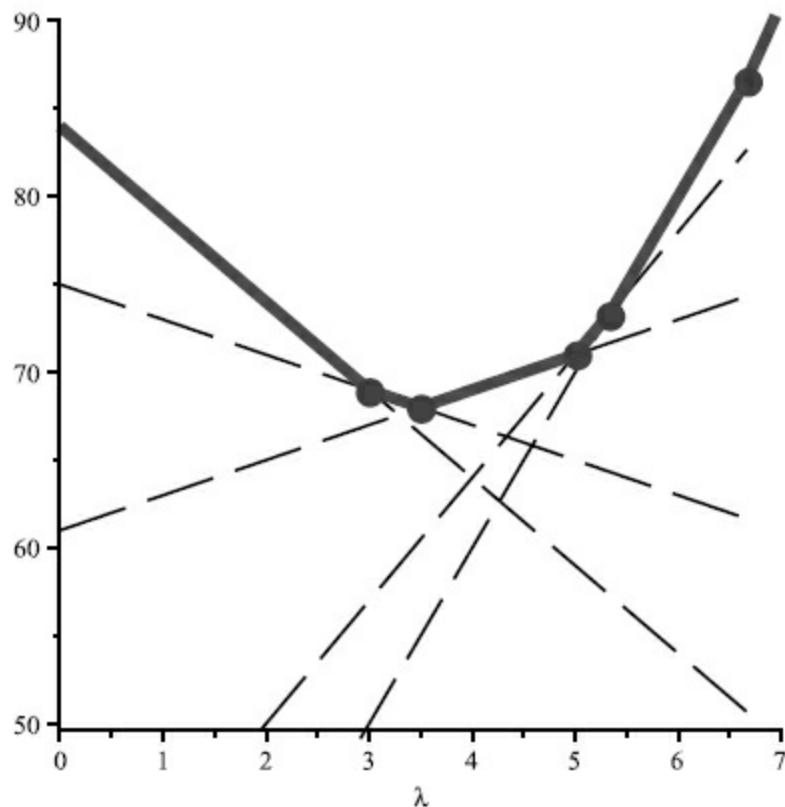
$$3x_1 + 3x_2 + 5x_3 + 4x_4 + 3x_5 \leq 13$$

$$x_i \in \{0, 1\}, \quad i = 1, \dots, 5.$$

By multiplying the knapsack constraint by $\lambda \geq 0$, and moving this constraint into the objective function, we get the Lagrangian relaxation

$$\begin{aligned} z_{\text{LD}}^* &= \min_{\lambda \geq 0} L(\lambda) \\ &= \min_{\lambda \geq 0} \max_{\mathbf{x} \in \{0,1\}^5} 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5 \\ &\quad + \lambda (13 - (3x_1 + 3x_2 + 5x_3 + 4x_4 + 3x_5)) \\ &= \min_{\lambda \geq 0} \left(13\lambda + \max_{\mathbf{x} \in \{0,1\}^5} (20 - 3\lambda)x_1 + (16 - 3\lambda)x_2 + (25 - 5\lambda)x_3 \right. \\ &\quad \left. + (14 - 4\lambda)x_4 + (9 - 3\lambda)x_5 \right). \end{aligned}$$

FIGURE 13.3 Graph of $L(\lambda)$ in Example 13.8.



It is easy to verify (see Exercise 13.8) that

$$L(\lambda) = \begin{cases} 13\lambda + (84 - 18\lambda), & 0 \leq \lambda \leq 3, \\ 13\lambda + (75 - 15\lambda), & 3 \leq \lambda \leq \frac{14}{4}, \\ 13\lambda + (61 - 11\lambda), & \frac{14}{4} \leq \lambda \leq 5, \\ 13\lambda + (36 - 6\lambda), & 5 \leq \lambda \leq \frac{16}{3}, \\ 13\lambda + (20 - 3\lambda), & \frac{16}{3} \leq \lambda \leq \frac{20}{3}. \end{cases}$$

[Figure 13.3](#) gives the graph of $L(\lambda)$ that indicates that the minimum value occurs at $\bar{\lambda} = \frac{14}{4}$ with $z_{LP} = 68$.

Why are there different relaxation approaches? From a computational standpoint, Lagrangian relaxations can be more efficiently solved for some problems, such as those from network flow problems with additional side constraints. In other cases, some relaxations can be resolved much more quickly if small changes to the problem are made; we will see such a case in the next section when we try to determine whether a variable's optimal value can be determined without completely solving the original problem. In any case, a relaxation gives us some information regarding the optimality of the current solution.

13.3 PREPROCESSING AND PROBING

The success of any algorithm for solving integer programs relies on both the formulation of our problem and the relaxation used to solve it. Modern software for solving integer programs includes techniques for logically improving the formulation as well as improving variable bounds derived from properties of the optimal solution. Such *preprocessing* and *probing* techniques are often simple rules that can have great influence on the performance of algorithms for integer programs. In this section, we shall discuss some of these basic approaches.

Preprocessing Modern optimization software packages will employ some sort of preprocessing to the problem to improve the formulation of the problem. This is done by improving bounds on variables and possibly

detecting redundancies in the constraints. These are typically based on very quick and simple ideas. The goal is to have a smaller problem that will solve more quickly. Example 13.9 provides an illustration of this idea.

■ EXAMPLE 13.9

Consider the linear program

$$\max \quad 2x_1 + 3x_2 + x_3$$

s.t.

$$8x_1 + 4x_2 - 3x_3 \leq 21$$

$$3x_1 - 2x_2 + 4x_3 \geq 16$$

$$0 \leq x_1 \leq 5$$

$$1 \leq x_2 \leq 4$$

$$0 \leq x_3 \leq 3.$$

Let's examine the first constraint. If we solve for x_1 , we get

$$8x_1 \leq 21 - 4x_2 + 3x_3.$$

Using the bounds on x_2 and x_3 generates the inequality

$$8x_1 \leq 21 - 4(1) + 3(3) = 26,$$

or $x_1 \leq \frac{26}{8} = \frac{13}{4}$. Repeating the same approach for x_2 and x_3 gives

$$4x_2 \leq 21 - 8x_1 + 3x_3 \leq 21 + 0 + 9 = 30$$

$$3x_3 \geq 8x_1 + 4x_2 - 21 \geq 0 + 4 - 21 = -17,$$

which does not improve upon any bounds. Considering the second constraint, we get the following inequalities:

$$3x_1 \geq 16 + 2x_2 - 4x_3 \geq 16 + 2 - 12 = 6$$

$$2x_2 \leq 3x_1 + 4x_3 - 16 \leq 3\left(\frac{13}{4}\right) + 4(3) - 16 = \frac{23}{4}$$

$$4x_3 \geq 16 - 3x_1 + 2x_2 \geq 16 - 3\left(\frac{13}{4}\right) + 2(1) = \frac{33}{4}.$$

Combining all of above information gives the improved bounds

$$\begin{aligned}2 &\leq x_1 \leq \frac{13}{4} \\1 &\leq x_2 \leq \frac{23}{8} \\\frac{33}{16} &\leq x_3 \leq 3.\end{aligned}$$

Note that if we also constrain the variables to be integer valued, we get the further improved bounds

$$\begin{aligned}2 &\leq x_1 \leq 3 \\1 &\leq x_2 \leq 2 \\x_3 &= 3.\end{aligned}$$

For binary integer programs, we can also look for more logical relations between the variables. These involve relating variables together in the hopes of fixing some of the variables to specific values or removing variables from the problem via equalities. Such an approach is again best illustrated via an example.

■ EXAMPLE 13.10

Consider the constraints

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

Notice that, in the first constraint, if $x_1 = 1$, then we must have $x_2 = 1$ in order to have a feasible solution. This relation can be expressed as $x_1 \leq x_2$ since the variables are both binary. In the second constraint, note that both x_2 and x_3 cannot have value 1 in a feasible solution; thus, we have $x_2 + x_3 \leq 1$. In the third constraint, note that at least two of the variables must have value 1, generating the relationship $x_2 + x_3 + x_4 \geq 2$. However, these last two relations imply that $x_4 = 1$, which then leads to $x_2 + x_3 = 1$. Putting together the constraints and these logical implications, we find that the only feasible solutions to this problem are $(0, 1, 0, 1)$, $(1, 1, 0, 1)$, and $(0, 0, 1, 1)$.

Preprocessing enables the user to solve larger problems in shorter time than without it. In fact, not-so-difficult-looking problems can become difficult to

solve without any preprocessing.

■ EXAMPLE 13.11

Let's consider the integer program

$$\begin{aligned} \min \quad & y \\ \text{s.t.} \quad & y + \sum_{j=1}^n 2x_j = n - 1 \\ & x_j, y \in \{0, 1\}. \end{aligned}$$

If n is an even number, then clearly the optimal value to this problem is 1. If we solve this problem for n as small as 30, we find that with preprocessing modern optimization software takes fractions of a second, but if we skip the preprocessing stage, the problem may take considerably longer (a few minutes longer even when $n = 30$).

Probing Another approach often taken to improve the formulation of an integer program tries other approaches to improve variable bounds, based on known feasible integer solutions and objective value bounds. Such *probing* approaches are best illustrated on 0–1 integer programs. Suppose we have a known (incumbent) feasible solution with value z_0 , and variable x_k whose value is 1 in this solution. We want to know if it is possible for x_k to have value 0 and still have (potentially) a feasible solution whose value is greater than z_0 . What we can do is fix $x_k = 0$ at the current subproblem and solve a relaxation. If the relaxed solution has value less than or equal to z_0 , we can fix $x_k = 1$. If this value is greater than z_0 , we do not know anything about the optimal value of x_k . A similar analysis holds true if we test $x_k = 0$.

■ EXAMPLE 13.12

Consider the following 0–1 knapsack problem:

$$\begin{aligned} \max \quad & 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5 \\ \text{s.t.} \quad & 3x_1 + 3x_2 + 5x_3 + 4x_4 + 3x_5 \leq 13 \\ & x_i \in \{0, 1\}, \quad i = 1, \dots, 5. \end{aligned}$$

The LP-relaxation solution is $(1, 1, 1, 0.5, 0)$ with value $z_{LP} = 68$. An initial feasible solution is $(1, 1, 1, 0, 0)$ with value 61. Suppose we wish to identify whether x_1 could be 0 in some integer solution whose value is more than 61. If we set $x_1 = 0$, we have the knapsack problem

$$\begin{aligned} \max \quad & 16x_2 + 25x_3 + 14x_4 + 9x_5 \\ \text{s.t.} \quad & 3x_2 + 5x_3 + 4x_4 + 3x_5 \leq 13 \\ & x_i \in \{0, 1\}, \quad i = 2, \dots, 5, \end{aligned}$$

whose optimal LP-relaxation solution is $(0, 1, 1, 1, \frac{1}{3})$, with value 58. Since $58 < 61$, we can surmise that in any solution to our problem with value at least 61, $x_1 = 1$. Hence, we can fix $x_1 = 1$ for every subsequent solution.

Unfortunately, the most obvious way to probe—solve the LP-relaxation after fixing x_k —typically takes too much time if it is done for each variable. One potential way to reduce this complexity is to not necessarily solve a relaxed problem, but find a bound to its value very quickly. To illustrate, consider the 0–1 knapsack problem

$$\begin{aligned} \max \quad & c_1x_1 + c_2x_2 + \dots + c_nx_n \\ \text{s.t.} \quad & a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b \\ & x_i \in \{0, 1\}, \quad i = 1, 2, \dots, n \end{aligned}$$

and consider its Lagrangian relaxation. If we let

$$L(\lambda) = \sum_{i=1}^n c_i x_i + \lambda \left(b - \sum_{i=1}^n a_i x_i \right),$$

then

$$\begin{aligned} z_{LD} &= \min_{\lambda \geq 0} \max_{\mathbf{x} \in \{0,1\}^n} L(\lambda) \\ (13.7) \quad &= \min_{\lambda \geq 0} \left(\lambda b + \max_{\mathbf{x} \in \{0,1\}^n} \sum_{i=1}^n (c_i - \lambda a_i)x_i \right). \end{aligned}$$

It is easy to see that for each fixed $\lambda \geq 0$, an optimal \mathbf{x} to (13.7) satisfies

$$x_k = \begin{cases} 1, & \text{if } c_i - \lambda a_i \geq 0, \\ 0, & \text{if } c_i - \lambda a_i < 0. \end{cases}$$

Furthermore, for each $\lambda \geq 0$ and for each \mathbf{x} feasible to the 0–1 knapsack problem,

$$\sum_{i=1}^n c_i x_i \leq \sum_{i=1}^n c_i x_i + \lambda \left(b - \sum_{i=1}^n a_i x_i \right).$$

This then implies that the optimal integer solution value z_{IP} to the 0–1 knapsack problem (13.7) satisfies $z_{\text{IP}} \leq z_{\text{LD}}$. In addition, in Exercise 9.18, we proved that $z_{\text{LD}} \leq z_{\text{LP}}$, where z_{LP} is the optimal value of the LP-relaxation to the 0–1 knapsack problem. The optimal solution to the LP-relaxation is

$$x_k = \begin{cases} 1, & \text{if } k < r, \\ 0, & \text{if } k > r, \\ \frac{b - \sum_{i=1}^{r-1} a_i}{a_r}, & \text{if } k = r, \end{cases}$$

where r is the unique index such that

$$(13.8) \quad \sum_{i=1}^{r-1} a_i \leq b < \sum_{i=1}^r a_i,$$

and its value is

$$\sum_{i=1}^{r-1} c_i + \frac{c_r}{a_r} \left(b - \sum_{i=1}^{r-1} a_i \right).$$

So how can we use the Lagrangian relaxation to fix variables? If we fix x_k for $k \neq r$, we can use (13.7) with $\lambda = \frac{\bar{c}_r}{a_r}$ and obtain an upper bound to z_{LD} very quickly; this value is equal to z_{LP} . For example, suppose $k < r$, and we want to test whether $x_k = 0$ in some integer solution whose value is larger than z_0 . By removing x_k from the original knapsack problem, an upper bound can be calculated from (13.7) as

$$\begin{aligned} \sum_{i=1}^{r-1} c_i - c_k + \frac{c_r}{a_r} \left(b - \sum_{i=1}^{r-1} a_i + a_k \right) &= \sum_{i=1}^{r-1} c_i + \frac{c_r}{a_r} \left(b - \sum_{i=1}^{r-1} a_i \right) - \left(c_k - \frac{c_r}{a_r} a_k \right) \\ &= z_{\text{LP}} - \left(c_k - \frac{c_r}{a_r} a_k \right). \end{aligned}$$

Hence, if we've already solved the LP-relaxation of our problem and find that

$$z_{\text{LP}} - \left(c_k - \frac{c_r}{a_r} a_k \right) \leq z_0,$$

we know $x_k = 1$ in an optimal integer solution to the problem. To test whether $x_k = 0$ in an optimal integer solution, where $k > r$, we again can rely on the Lagrangian relaxation to find an upper bound when $x_k = 1$:

$$\begin{aligned} \sum_{i=1}^{r-1} c_i + c_k + \frac{c_r}{a_r} \left(b - \sum_{i=1}^{r-1} a_i - a_k \right) &= \sum_{i=1}^{r-1} c_i + \frac{c_r}{a_r} \left(b - \sum_{i=1}^{r-1} a_i \right) - \left(\frac{c_r}{a_r} a_k - c_k \right) \\ &= z_{\text{LP}} - \left(\frac{c_r}{a_r} a_k - c_k \right). \end{aligned}$$

Hence, we have the following rules:

Variable Fixation Rule 1. For $k < r$, where r is the unique index satisfying (13.8),

we can fix $x_k = 1$ in our 0–1 knapsack problem if

$$z_{\text{LP}} - z_0 \leq c_k - \frac{c_r}{a_r} a_k.$$

Variable Fixation Rule 2. For $k > r$, where r is the unique index satisfying (13.8),

we can fix $x_k = 0$ in our 0–1 knapsack problem if

$$z_{\text{LP}} - z_0 \leq \frac{c_r}{a_r} a_k - c_k.$$

■ EXAMPLE 13.13

Consider again the knapsack problem

$$\begin{aligned} \max \quad & 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5 \\ \text{s.t.} \quad & 3x_1 + 3x_2 + 5x_3 + 4x_4 + 3x_5 \leq 13 \\ & x_i \in \{0, 1\}, \quad i = 1, \dots, 5. \end{aligned}$$

This problem has an LP-relaxation solution of $(1, 1, 1, 0.5, 0)$ with value $z_{LP} = 68$. Note that $r = 4$ and that $\frac{c_r}{a_r} = \frac{14}{4}$. An initial feasible solution is $(1, 1, 1, 0, 0)$ with value 61. To check whether x_1 can be fixed to 1, we use variable fixation rule 1:

$$x_1 : \text{Is } 68 - 61 \leq 20 - \frac{14}{4}(3) ?$$

Since the answer is YES, we can fix $x_1 = 1$. Here are the questions that need to be asked to check on variables x_2, x_3, x_5 :

$$x_2 : \text{Is } 68 - 61 \leq 16 - \frac{14}{4}(3) ?$$

$$x_3 : \text{Is } 68 - 61 \leq 25 - \frac{14}{4}(5) ?$$

$$x_5 : \text{Is } 68 - 61 \leq \frac{14}{4}(3) - 9 ?$$

Since the only YES answer to the above questions is for x_3 , we can fix variable $x_3 = 1$ as well. This reduces our original 0–1 knapsack problem to

$$\begin{aligned} & 45 + \max \quad 16x_2 + 14x_4 + 9x_5 \\ & \text{s.t.} \\ & \quad 3x_2 + 4x_4 + 3x_5 \leq 5 \\ & \quad x_i \in \{0, 1\}, \quad i = 2, 4, 5. \end{aligned}$$

It is now easy to see that the optimal solution is $(1, 1, 1, 0, 0)$ with value 61. As this example shows, probing can greatly simplify our problem, allowing us to solve large problems with less work.

13.4 WHEN ARE INTEGER PROGRAMS “EASY?”

As noted in the introduction to this chapter, integer programs, in general, are much more difficult to solve than linear programs; in fact, the existence of an algorithm that solves every integer program efficiently probably does not exist. We often call such problems **intractable**, since the computational time to solve some instances can be extremely long relative to the size of the

problem. That being said, there are cases or specific subclasses of integer programs (and general discrete optimization problems) for which efficient algorithms to solve them exist or, more simply, whose LP-relaxation always generates an integer solution. In fact, a common research direction is to find such “easy” subclasses of problems. In this section, we introduce some of these classes of problems.

Totally Unimodular Matrices We have already seen one instance of an easy class of integer problems in Section 13.1.

Lemma 13.1 *Suppose we have an integer program*

$$\begin{aligned}
 & \max \quad \mathbf{c}^T \mathbf{x} \\
 \text{s.t.} \\
 & A\mathbf{x} \leq \mathbf{b} \\
 (13.9) \quad & \mathbf{x} \geq \mathbf{0}, \text{ integer,}
 \end{aligned}$$

where every element of A and \mathbf{b} are integers. If every extreme point of

$$P = \{\mathbf{x} \geq \mathbf{0} : A\mathbf{x} \leq \mathbf{b}\}$$

contains only integer coordinates, then solving the LP-relaxation of (13.9) by the simplex method produces a solution to (13.9).

In fact, if we have an ideal formulation of the feasible integer solutions to our integer program, then, since linear programs can be solved efficiently, we can solve the integer program efficiently. The problem is that, in general, we can never be sure if our formulation is ideal.

To find a problem whose LP-relaxation is an ideal formulation, consider an integer program whose LP-relaxation is in canonical form

$$\begin{aligned}
 & \max \quad \mathbf{c}^T \mathbf{x} \\
 \text{s.t.} \\
 & A\mathbf{x} = \mathbf{b} \\
 & \mathbf{x} \geq \mathbf{0}, \text{ integer.}
 \end{aligned}$$

Recall that if an optimal solution to the LP-relaxation exists, then at least one of the optimal solutions is a basic feasible solution. Such a solution can be written in the form

$$\mathbf{x}^* = \begin{bmatrix} \mathbf{x}_B^* \\ \mathbf{x}_N^* \end{bmatrix} = \begin{bmatrix} B^{-1}\mathbf{b} \\ \mathbf{0} \end{bmatrix},$$

where B is a nonsingular $m \times m$ submatrix of A . Thus, \mathbf{x}_B^* is the solution to the matrix equation $B\mathbf{x} = \mathbf{b}$. One characterization of such a solution to a matrix equation is Cramer's rule.

Lemma 13.2 (Cramer's Rule) *Given a matrix equation $A\mathbf{x} = \mathbf{b}$, where A is non-singular,*

$$x_k = \frac{\det(A_k^*)}{\det(A)},$$

where A_k^* is the matrix formed from A and \mathbf{b} by replacing the k th column of A by \mathbf{b} .

■ EXAMPLE 13.14

Suppose we wish to solve the system of equations

$$\begin{aligned} 3x + 2y &= 11 \\ x + 4y &= 7 \end{aligned}$$

using Cramer's rule. Note that

$$\det(A) = \det \begin{pmatrix} 3 & 2 \\ 1 & 4 \end{pmatrix} = 10.$$

Thus, we have our solution

$$\begin{aligned} x &= \frac{\det \begin{pmatrix} 11 & 2 \\ 7 & 4 \end{pmatrix}}{\det(A)} \\ &= \frac{30}{10} = 3 \\ y &= \frac{\det \begin{pmatrix} 3 & 11 \\ 1 & 7 \end{pmatrix}}{\det(A)} \\ &= \frac{10}{10} = 1. \end{aligned}$$

Returning to our optimal basic feasible solution, since B is an integer matrix and \mathbf{b} an integer vector, the determinant $\det(B_k^*)$ in Cramer's rule will always be an integer. Thus, we get the following result.

Lemma 13.3 *Given a linear program in canonical where the constraint matrix A and the vector \mathbf{b} have all integer components, every basic feasible solution has all integer components if every $m \times m$ nonsingular submatrix B*

has determinant either 1 or -1 .

Note that not all linear programs have constraint matrices that satisfy this property.

■ EXAMPLE 13.15

Consider the linear program (in canonical form)

$$\begin{aligned} \max \quad & z = 13x + 5y \\ \text{s.t.} \quad & \\ & 4x + y + s_1 = 24 \\ & x + 3y + s_2 = 24 \\ & 3x + 2y + s_3 = 23 \\ & x, y, s_1, s_2, s_3 \geq 0. \end{aligned}$$

We've previously seen that every basic feasible solution has only integer components. Now, consider the basic feasible solution $(6, 0, 0, 18, 5)$ with basis $\mathcal{B} = \{x, s_2, s_3\}$ to the problem in canonical form. We then have

$$\det(B) = \det \begin{pmatrix} 4 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} = 4.$$

Hence, Lemma 13.3 does not completely characterize those linear programs that have all-integer basic feasible solutions.

Now consider integer programs in the form (13.9). Recall that every extreme point is found at the intersection of at least n active constraints/bounds. We can write this as the matrix equation

$$D\mathbf{x} = \mathbf{d},$$

where the matrix D contains some rows of A and some rows from the active bounds $x_k = 0$. Since we don't know exactly how many rows of A will be included, we need a more general result than that of Lemma 13.3. We begin with the following definition.

Totally Unimodular Matrix A matrix A is *totally unimodular* if every square submatrix of A has determinant $+1$, -1 , or 0 .

Note that if a matrix A is totally unimodular, then every element must be 0 , 1 ,

or -1 . Again, this does not completely characterize such matrices.

■ EXAMPLE 13.16

It is easy to see that the matrix

$$A = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

is not totally unimodular, while the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

is totally unimodular.

Totally unimodular matrices were discovered in the 1950s as people tried to determine when a polyhedron $P = \{ \mathbf{x} \geq \mathbf{0} : A\mathbf{x} \leq \mathbf{b} \}$ has all-integer extreme points. The following result is due to Hoffman and Kruskal [59].

Let A be an matrix with all integer components. The polyhedron $P = \{ \mathbf{x} \geq \mathbf{0} : A\mathbf{x} \leq \mathbf{b} \}$ has all-integer extreme points for **every** integer vector \mathbf{b} if and only if A is totally unimodular.

At this point, a natural question to ask is “Which matrices are totally unimodular?” Various characterizations have been given over the years. Below are some results related to totally unimodular matrices.

Lemma 13.4 *A matrix A is totally unimodular if and only if A^T is totally unimodular.*

Lemma 13.5 *A matrix A is totally unimodular if and only if the matrix $[A|I]$ is totally unimodular.*

Lemma 13.6 *A matrix A is totally unimodular if*

1. $a_{ij} \in \{ 0, 1, -1 \}$ for all i, j .
2. *Each column contains at most two nonzero elements.*
3. *There exists a partition (M_1, M_2) of the rows of A such that for each column j containing exactly two nonzero elements,*

$$\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0.$$

■ EXAMPLE 13.17

Consider the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

Clearly, it satisfies the first two conditions of Lemma 13.6. To check the third, consider the partition $M_1 = \{1, 3\}$, $M_2 = \{2\}$ of the rows. To check condition 3, we need to check only columns 1 and 3. Column 1 gives $(1 + (-1)) - (0) = 0$ and column 3 gives $(1 + 0) - (1) = 0$. Hence, A is totally unimodular.

Network Flow Problems While totally unimodular constraint matrices guarantee that integer program can easily be solved by solving its LP-relaxation, at this point it is unclear whether this is anything more than a nice theoretical result with no real applicable (or common) optimization problems satisfying this property. Fortunately, this is not the case.

Recall the network optimization problems introduced in Chapters 2 and 12: Each of these problems can be formulated as the optimization model

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j) \in A} x_{ij} - \sum_{k:(k,i) \in A} x_{ki} = b(i), \quad i \in V \\ & 0 \leq x_{ij} \leq u_{ij}, \quad (i, j) \in A. \end{aligned}$$

Note that we must have $\sum_{i \in V} b(i) = 0$ to have a feasible solution.

The shortest path problem, the maximum flow problem, and the minimum cost network flow problem can each be formulated as integer programs if we want integer flow values. We saw in Chapter 12 that each problem has an integer solution if all data are integer valued. While we did not state it in this manner, the following provides further justification for this result.

The constraint matrix A in the minimum cost network flow problem is totally unimodular.

For these network flow problems, having integer-valued optimal solutions is just the start. Specialized algorithms taking advantage of the combinatorial nature of the networks have been developed that solve these problems both theoretically and computationally more efficiently than solving them as linear programs; we saw examples of these in Chapter 12. For more information on such algorithms, the reader is urged to see books by Ahuja, Magnanti, and Orlin [2] and Bertsekas [13].

Summary

In this chapter, we introduced some of the basic terms related to integer programs and explored the importance of formulating our problem as “ideally” as possible. We saw how optimization packages use preprocessing and probing to reduce the size of our problem, which is important given that integer programs can be computationally inefficient to solve. Now that we have discussed these basic formulation issues, we can turn our attention to actually solving these problems. This is the basis for Chapter 14.

EXERCISES

13.1 Find a better formulation for

$$X = \left\{ \mathbf{x} \in \{0, 1\}^5 : 3x_1 + 2x_2 + 5x_3 + 4x_4 + 2x_5 \leq 13 \right\}.$$

13.2 Find an ideal formulation of

$$X = \left\{ (x, y) : \begin{array}{l} -18x + 38y \leq 133 \\ 13x + 11y \leq 125 \\ 10x - 8y \leq 55 \\ x, y \geq 0, \text{ integer} \end{array} \right\}.$$

13.3 Find an ideal formulation of

$$X = \left\{ (x, y) : \begin{array}{l} x - y \geq -1 \\ 3x + 2y \leq 3 \\ 6x + y \leq 14 \\ x, y \geq 0, \text{ integer} \end{array} \right\}.$$

13.4 Verify that $P_2 \subset P_1$ in Example 13.4.

13.5 In a fixed-charge problem, given a collection of variables $x_k \in \{0, 1\}, k = 1, 2, 3$, we want variable $y = 1$ if at least one of the x_k 's is equal to 1. Two formulations of this are

$$P_1 = \{x_1, x_2, x_3, y \in \{0, 1\} : x_1 + x_2 + x_3 \leq 3y\},$$

$$P_2 = \left\{ x_1, x_2, x_3, y \in \{0, 1\} : \begin{array}{l} x_1 \leq y \\ x_2 \leq y \\ x_3 \leq y \end{array} \right\}.$$

Show that P_2 is a better formulation than P_1 .

13.6 Use preprocessing on the following integer program to simplify its formulation.

$$\begin{aligned} \max \quad & x_1 + 2x_2 + x_3 \\ \text{s.t.} \quad & 6x_1 - 2x_2 + 9x_3 \leq 16 \\ & 9x_1 + 4x_2 - 2x_3 \geq 6 \\ & x_1 + x_2 + x_3 \leq 7 \\ & 0 \leq x_1 \leq 3 \\ & 0 \leq x_2 \leq 1 \\ & 1 \leq x_3 \leq 3. \end{aligned}$$

13.7 Use the probing techniques of Section 13.3 to fix some of the variables in the problem

$$\begin{aligned} \max \quad & 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5 \\ \text{s.t.} \quad & 3x_1 + 2x_2 + 5x_3 + 4x_4 + 2x_5 \leq 13 \\ & x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}. \end{aligned}$$

13.8 Verify the calculation of $L(\lambda)$ and z_{LD} in Example 13.8.

13.9 Formulate and solve the Lagrangian relaxation of the knapsack problem

$$\begin{aligned} \max \quad & 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5 \\ \text{s.t.} \quad & \end{aligned}$$

$$3x_1 + 2x_2 + 5x_3 + 4x_4 + 2x_5 \leq 13$$

$$x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}.$$

13.10 Using a optimization package, turn off the preprocessing and solve the integer program given in Example 13.11 for $n = 35, 40, 45, 50$. Compare the running times for each value of n . How fast is the rate of increase of this time?

13.11 Show that the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 \end{bmatrix}$$

is totally unimodular using Lemma 13.6. Note that this is the constraint matrix for a transportation problem with two supply nodes and three demand nodes.

13.12 Show that the matrix

$$A = \begin{bmatrix} 1 & 0 & 1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 \end{bmatrix}$$

is totally unimodular using Lemma 13.6.

CHAPTER 14

SOLVING INTEGER PROGRAMS: EXACT METHODS

In Chapter 13, we began our discussion of integer programs by first considering how to formulate them and some ways, such as preprocessing and probing, in which we can improve these formulations. We also mentioned that such concerns were relevant since, unlike linear programs, integer programs did not have an optimality condition associated with it, which means that, even if we have obtained the optimal solution, we cannot verify this fact until we have essentially “explored” all other possible solutions. It is this exploration that can be time-consuming. In this chapter, we examine various approaches for solving integer programs and discuss methods that are currently utilized in optimization software packages.

14.1 COMPLETE ENUMERATION

Out of all methods for solving integer programs, perhaps the most basic method, at least from a theoretical perspective, is to do a *complete enumeration*, in which we explicitly find each feasible integer solution, check its objective function value to see if it is the current best solution or not, and proceed to the next solution.

■ EXAMPLE 14.1

Suppose we consider the integer program

$$\max \left\{ 100x_1 + 50x_2 + 60x_3 + 45x_4 : (x_1, x_2, x_3, x_4) \in P_1 \cap \mathbb{Z}^4 \right\},$$

where

$$P_1 = \left\{ \mathbf{x} : \begin{array}{l} 83x_1 + 57x_2 + 40x_3 + 19x_4 \leq 100 \\ 0 \leq x_i \leq 1, \quad i \in \{1, \dots, 4\} \end{array} \right\}.$$

In Example 13.3, we saw that the set of solutions in $P_1 \cap \mathbb{Z}^4 = X$, where

$$X = \left\{ \begin{array}{l} (0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), \\ (0, 0, 0, 1), (0, 1, 1, 0), (0, 1, 0, 1), (0, 0, 1, 1) \end{array} \right\}.$$

By explicitly checking each of these solutions, we find that the optimal solution is $(0, 1, 1, 0)$ with optimal value $z^* = 110$.

Sounds simple, doesn't it? Unfortunately, even if the number of variables is relatively small, the number of integer solutions we would have to check becomes very large. This is often referred to as the "combinatorial explosion" of integer problems.

■ EXAMPLE 14.2

Consider the following 0–1 knapsack problem

$$\max \quad \sum_{i=1}^n c_i x_i$$

s.t.

$$\sum_{i=1}^n a_i x_i \leq b$$

$$x_i \in \{0, 1\}, i = 1, 2, \dots, n,$$

where each $c_i, a_i > 0$. Suppose that $b = \frac{1}{2} \sum_{i=1}^n a_i$. If we disregard the constraint, there would be 2^n possible 0–1 vectors that need to be checked. When we consider the constraint, roughly half of these solution vectors remain feasible, so approximately 2^{n-1} solutions need to be checked. If we have $n = 100$ variables, that means we need to check $2^{99} \approx 1.27 \times 10^{30}$ possible solutions. If we can check 1 billion solutions per second, we'd need to spend roughly 10^{21} seconds or about 3.1×10^{13} years just to solve this problem.

Of course, as exemplified by the above example, complete enumeration is only practical if there are a very small number of variables. What would be

nice is to implicitly enumerate all possible solutions, where we determine in the middle of our algorithm that certain solutions need not be considered since we know that they cannot be the optimal solution. This is the idea behind our next approach.

14.2 BRANCH-AND-BOUND METHODS

When we attempt to solve an integer program, it would be nice to eliminate as many of the possible solutions as we can without actually evaluating them. One approach is to divide the feasible region into smaller subregions and evaluate each of the smaller subregions individually. This *divide and conquer* approach is based upon the following idea.

Consider the optimization problem $z^* = \max\{f(\mathbf{x}) : \mathbf{x} \in S\}$, where S is the feasible region. If S is decomposed into the subregions $S = S_1 \cup S_2 \cup \dots \cup S_K$ and $z_k = \max\{f(\mathbf{x}) : \mathbf{x} \in S_k\}$ is the optimal value over each of the subregions, $k = 1, 2, \dots, K$, then $z^* = \max\{z_k : k = 1, 2, \dots, K\}$.

Of course, solving the optimization problems over the smaller subregions of S may not be any easier than over the original set S . Suppose, instead, that we know an upper bound to the optimal value z_k of the k th subregion, that is, a value $\bar{z}_k \geq z_k$. If we have an integer solution \mathbf{x}^0 in S with objective function value z_0 where $z_0 \geq \bar{z}_k$, then we know that the only way a feasible solution in subregion S_k can possibly be optimal is if it has value z_0 . In this case, we do not need to consider any of these feasible solutions in S_k , since we already have a feasible solution with value z_0 , and we don't need another (unless, of course, we're trying to find all optimal solutions). Therefore, we can eliminate from consideration all feasible solutions in S_k , and hence reduce the amount of work we need to do. This is the basis for *branch-and-bound algorithms*, which are the most commonly used methods for solving integer programs.

Branch-and-Bound Algorithms attempt to solve discrete optimization problems by dividing the feasible region and examining each subregion for the best integer solution. It often uses a relaxed version of the problem to obtain bounds on the integer solution as a means of determining whether a subregion is worth investigating.

When we use a branch-and-bound approach, many details need to be finalized. For example, how can we get an upper bound, how do we find a feasible solution, or, perhaps most importantly, how do we partition the feasible region into subregions.

Obtaining an Upper Bound Let's first consider the upper bound. We saw in Section 13.2 the definition of a *relaxation*, which is an optimization problem where the original feasible region is a subset of the relaxation's feasible region. In particular, if we can solve a relaxation of a maximization problem, then it provides an upper bound to the original problem. Since we can form the linear programming relaxation very easily by removing the integrality constraints, and we can solve linear programs easily, these seem to be a natural choice to obtain an upper bound. Note that the LP-relaxation is not the only choice, but also is the one most often used.

Partitioning the Feasible Region Our next question is how to partition the feasible region. Determining a good decomposition seems to be just as hard as the original problem. One approach that has worked well is to decompose the feasible region in steps. The following example illustrates this concept.

■ EXAMPLE 14.3

Let's consider again the integer program

$$\begin{aligned}
 z_{IP}^* = \max \quad & 8x + 7y \\
 \text{s.t.} \quad & -18x + 38y \leq 133 \\
 & 13x + 11y \leq 125 \\
 & 10x - 8y \leq 55 \\
 & x, y \geq 0, \text{ integer.}
 \end{aligned} \tag{14.1}$$

We saw in Example 13.7 that the linear programming relaxation gives an optimal value of $z_{LP}^* = 78.25$, which comes from the solution $x = 4.75$, $y =$

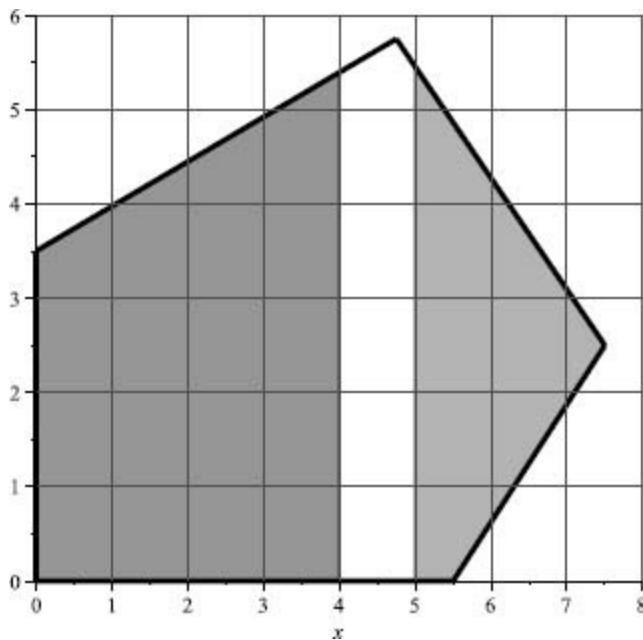
5.75. If we consider variable x , we can easily see that its optimal value must be either $x \leq 4$ or $x \geq 5$. Using this approach, we have started to decompose our feasible region (for the integer program) into those regions satisfying either $x \leq 4$ or $x \geq 5$. At this point, we can start all over again. While this may have appeared trivial, it has the effect of eliminating from consideration a section of the feasible region. For example, if we are to consider only those regions where either $x \leq 4$ or $x \geq 5$, then we would only be examining the shaded regions shown in [Figure 14.1](#). Note that the nonshaded region does not contain any integer solutions, and hence does not need to be examined. What we did in Example 14.3 is (1) solve a relaxed version of our problem, (2) find a variable x_k that does not have an integral value ($= v_k$) in the optimal relaxed solution, and (3) decompose the problem into subregions. Of course, it is possible that when we decompose our feasible region into the regions $x_k \leq \lfloor v_k \rfloor$ and $x_k \geq \lceil v_k \rceil$, we still do not have an easily identifiable integer solution. At this point, we would consider each of the subregions by going through the same process.

Branching and Branching Variables When we divide the current subregion into smaller ones in a branch-and-bound algorithm, we are said to be *branching*, and the variable with which we are decomposing the region is known as the **branching variable**.

Branch-and-Bound Tree A binary tree describing this decomposition is called the *branch-and-bound tree*. Each node in this tree corresponds to a relaxed subproblem obtained by the decomposition, while each edge corresponds to how this subproblem was transformed into the new subproblem. Often, this is due to the addition of a new constraint.

Root Problem and Node The initial relaxed problem is often referred to *root problem*, corresponding to the *root node*.

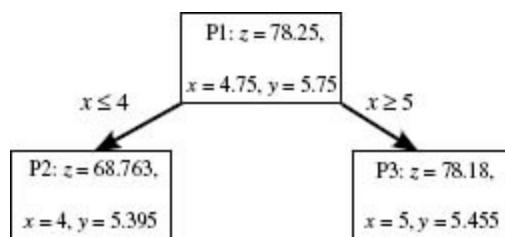
[FIGURE 14.1](#) Partitioning of feasible region in Example 14.3.



■ EXAMPLE 14.4

Considering again the integer program (14.1), if we solve the LP-relaxation, and choose x as our branching variable, we can construct a search tree that corresponds to the subproblems, such as the one given in [Figure 14.2](#). In problem P2, we assume that $x \leq 4$, while in subproblem P3 we assume that $x \geq 5$. Often the objective function value and the actual solution of the relaxed problem are kept within the corresponding node as a reference.

[**FIGURE 14.2**](#) Partial branch-and-bound tree for Example 14.4.



Note that when doing this “online” decomposition, it is possible to define a subproblem for which there are no feasible solutions. For example, suppose we branch on $x \geq 5$, and then we branch on $y \geq 6$. You should note that the third constraint of (14.1) cannot be satisfied in this subproblem. Does this cause a problem? No, we can automatically eliminate this subproblem.

In addition, we may come across a subproblem whose relaxed solution is integer valued. In such a case, we have a potentially optimal solution to the

original integer program. This **candidate solution** is then compared with the current best-known integer solution. If our new solution has a better objective function value, we store this new solution. Thus, at any subsequent step in the algorithm, we have a potential optimal integer solution at our disposal; this integer solution is often referred to as the **incumbent solution**. At this point, we no longer have to partition this subproblem, since we've found the best integer solution in this region.

What we have seen is that there are three ways in which a subproblem will not be considered for branching:

1. If the current subproblem yields an integer solution when the relaxed problem is solved.
2. When the current subproblem is infeasible.
3. When the current subproblem yields a relaxation bound that is not better than the value of the incumbent solution.

Fathom a Node We are said to *fathom* a subproblem (node) or *prune* the branch-and-bound tree if we do not branch from that subproblem due to either its infeasibility or because a bound on its optimal integer value is not better than the value of the current best-known integer solution to the original problem.

Active Nodes Those subproblems (nodes) that we have not fathomed yet or branched upon are called *active*.

When there are no active nodes, the current incumbent solution is the optimal solution to our problem. If there is no incumbent solution, then the integer program is infeasible.

It is this fathoming of a node that makes branch-and-bound methods better than complete enumeration. By fathoming, we are implicitly exploring a subproblem and concluding that an integer solution that is better than our current best-known integer solution does not exist in this subregion. What this also implies is that it is vitally important to find a good feasible integer solution as quickly as possible, so that we can eliminate many subproblems from consideration.

Choosing a Subproblem from Which to Branch Another issue we need to address when implementing a branch-and-bound algorithm is how to

determine which subproblem is examined next. Since we are dealing with a search tree (although one where each node is not explicitly given from the beginning), we can use some rules from binary search trees to choose our next subproblem. However, these do not have anything to do with the actual solutions to the subproblems themselves. In this thinking, two common ways to choose our next subproblem to examine (and branch from) are

1. *Best-First Search.* Here, we choose the subproblem whose relaxed solution has the best value (largest if a maximization problem, smallest if a minimization problem) among all those subproblems we have not branched from. This is because if we were to obtain a feasible solution in this subregion, we hope that its value would be very close to the generated upper bound. While there is no guarantee that this will occur, it seems to be a reasonable hope. However, one potential drawback to this approach is that we may need to store a large number of subproblems at any one time.
2. *Depth-First Search.* In this approach, we explore each subregion/node as if we were doing a depth-first search on the tree. In terms of the subproblems, this means that we examine one region completely before we proceed to examine any other. While this has some advantages, such as ease of use and the fact that initial feasible solutions are typically obtained more quickly, this tends to be the approach that takes the longest in terms of computational time. However, this works well in combination with best-first search because we can use depth-first search to find an initial feasible point (which allows for pruning), and then use best-first search to complete the optimization.

■ EXAMPLE 14.5

Continuing from Example 14.4, we see that, by restricting $x \leq 4$, our linear programming relaxation yields an upper bound of approximately 69.67, while restricting $x \geq 5$ yields an upper bound of 78.18. It seems reasonable to branch upon the second subproblem, since we potentially could have an integer solution with value as high as 78. In this case, our branching variable would be y , and our new subproblems would be where $y \leq 5$ and $y \geq 6$.

It is time to actually combine these discussions into a generic version of a branch-and-bound algorithm, given in Algorithm 14.1.

Algorithm 14.1 Generic Branch-and-Bound Algorithm

Step 0: (Initial Solution) Solve the problem above as a linear program (or another relaxation), ignoring the integrality restrictions. If all $x_i, i \in S$, have integral values, we are done—current solution is optimal. Otherwise, go to Step 1.

Step 1: (Branching Variable Selection) Choose, from among those variables $x_i, i \in S$, that do not have integral values at this node, one variable to be the branching variable. Suppose we choose variable x_k , with value v_k .

Step 2: (Formulation of New Nodes) Create two new mixed-integer problems represented by the node considered in Step 1. One problem adds the constraint

$$x_k \leq \lfloor v_k \rfloor$$

to the problem, while the other adds the constraint

$$x_k \geq \lceil v_k \rceil$$

to the problem. Solve each of these problems as a linear programming problem (or using another relaxation).

Step 3: (Test for Integer Solution) For each of the nodes created in Step 2, check if the corresponding solution is integer valued. If no, go to Step 4. If yes, we have a feasible solution to our mixed-integer problem. Compare the objective function value of this solution to that of our current best solution. If it is better, then save the current solution as our incumbent solution. Also, for every node not yet branched from, check its relaxed problem value to the new incumbent solution's value. If the relaxed bound is not better, then fathom the node. Go to Step 4.

Step 4: (Check Bound Against Incumbent Solution) If there is no incumbent solution, go to Step 6. Otherwise, for each node created in Step 2, check to see if the bound obtained by solving the relaxed problem is better than the objective function value of the current incumbent solution. If not, fathom that node.

Step 5: (Test for Infeasibility) For each of the nodes created in Step 2, check if the corresponding problem is infeasible. If yes, fathom that node.

Step 6: (Node Selection) Of all the nodes that are not fathomed and do not have a corresponding feasible solution to our mixed-integer problem (i.e., there exists a variable $x_i, i \in S$, that does not have integral value), choose one of these nodes to examine further. Go to Step 1. If there are no more nodes that need to be examined further, STOP. We have examined all possibilities. Our current best solution is the optimal solution.

■ EXAMPLE 14.6

Let's examine the integer program (14.1) first given in Example 14.3. We'll use the best-first approach to selecting a node to branch upon. Solving the root problem, we found that $x = 4.75$, $y = 5.75$, and $z = 78.25$. Since we do not have an integer solution, we select a variable to branch upon. Here, we'll select x , giving us the following subproblems:

P2 ($x \leq 4$) : $z = 69.763$, $x = 4$, $y = 5.395$

P3 ($x \geq 5$) : $z = 78.18$, $x = 5$, $y = 5.455$.

Neither of these problems yields an integer solution, so we choose problem P3 to branch from, with branching variable y . This yields the following subproblems:

$$P4 (x \geq 5, y \leq 5) : z = 78.077, x = 5.385, y = 5$$

$$P5 (x \geq 5, y \geq 6) : \text{INFEASIBLE}.$$

Hence, we can fathom node P5. Again, selecting node P4 to branch from, with branching variable x , we get the following subproblems:

$$P6 (x = 5, y \leq 5) : z = 75, x = 5, y = 5$$

$$P7 (x \geq 6, y \leq 5) : z = 77.909, x = 6, y = 4.273.$$

Note that, at this point, we have an incumbent solution of (5, 5). Since its value is larger than the upper bound found for subproblem P2, we fathom this node. However, we still need to proceed with subproblem P7, from which we next branch on y . This gives the following subproblems:

$$P8 (x \geq 6, y \leq 4) : z = 77.846, x = 6.231, y = 4$$

$$P9 (x \geq 6, y = 5) : \text{INFEASIBLE}.$$

Hence, we can fathom node P9. Since P8 is the only active node, it yields the following subproblems:

$$P10 (x = 6, y \leq 4) : z = 76, x = 6, y = 4$$

$$P11 (x \geq 7, y \leq 4) : z = 77.636, x = 7, y = 3.091.$$

Subproblem P10 yields an integer solution that has higher value than that of our incumbent solution, so the solution (6, 4) is our new incumbent. Again, there is only one active node (P11), so we get the following subproblems:

$$P12 (x \geq 7, y \leq 3) : z = 77.615, x = 7.077, y = 3$$

$$P13 (x \geq 7, y = 4) : \text{INFEASIBLE}.$$

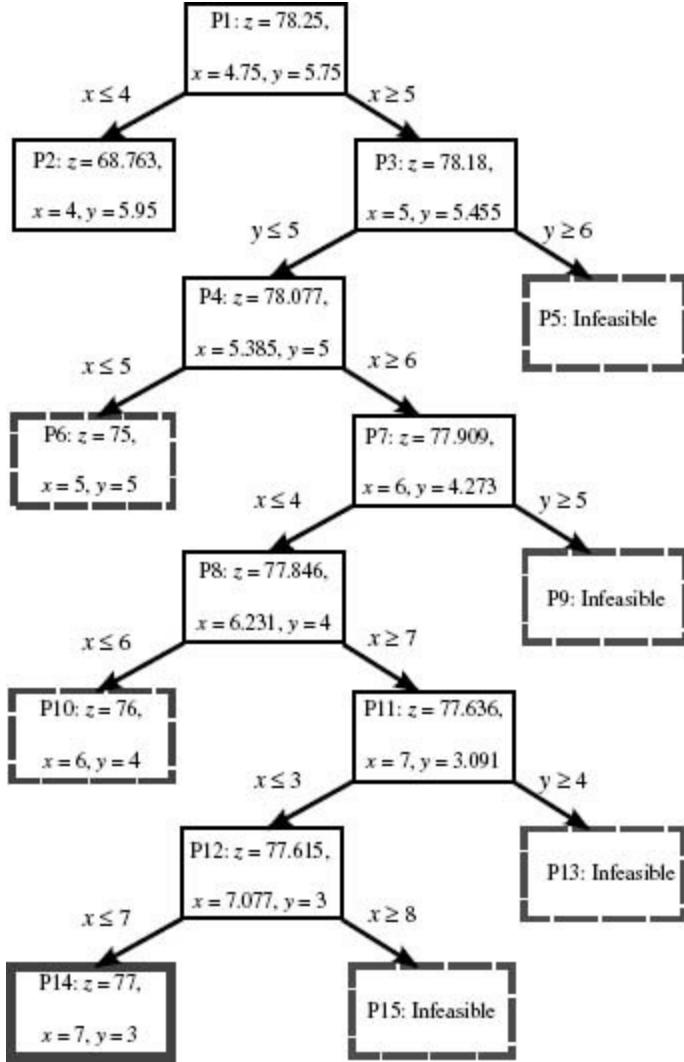
Again, we fathom node P13, leaving P12 as the only active node. This generates the following subproblems:

$$P14 (x = 7, y \leq 3) : z = 77, x = 7, y = 3$$

$$P15 (x \geq 8, y \leq 3) : \text{INFEASIBLE}.$$

Note that (7, 3) has larger value than our incumbent; hence, it becomes the incumbent solution. Furthermore, we fathom node P15, leaving us with no active nodes. This implies that the optimal solution to our integer program is (7, 3) with value = 77. In [Figure 14.3](#), we have the complete branch-and-bound tree for this problem.

FIGURE 14.3 Branch-and-bound tree for Example 14.6.



Of course, since integer programs are very difficult to solve this approach does not always work well. Because of this, other approaches have been proposed over the years. In these, more work is done before branching to determine which branching variable holds the most “promise” in terms of quicker solution times. In Section 14.7, we mention a few currently implemented methods.

14.3 VALID INEQUALITIES AND CUTTING PLANES

In Chapter 13, we introduced the notion of a formulation and talked about an

ideal formulation for an integer program, where every extreme point has only integer coordinates. We not only saw that the convex hull of all feasible integer solutions is an ideal formulation but also noted that finding every constraint of the convex hull can be computationally intractable, since there may be exponentially many of them. However, it is reasonable to believe that we can find better formulations of the feasible integer solutions, especially ones that provide better bounds. To do this, we need to determine additional constraints that are satisfied by all integer solutions to our original constraints.

Valid Inequality A *valid inequality* for a set of solutions X is an inequality that is satisfied by all solutions in X .

Introducing valid inequalities into a relaxation (especially an LP-relaxation) will typically improve the formulation obtained. However, there is a catch. Not every valid inequality improves the formulation.

■ EXAMPLE 14.7

Consider the set of solutions

$$X = \left\{ \mathbf{x} \in \mathbb{Z}^4 : \begin{array}{l} 83x_1 + 57x_2 + 40x_3 + 19x_4 \leq 100 \\ 0 \leq x_i \leq 1 \end{array} \right\}.$$

Its relaxed feasible region would be

$$P_1 = \left\{ \mathbf{x} \in \mathbb{R}^4 : \begin{array}{l} 83x_1 + 57x_2 + 40x_3 + 19x_4 \leq 100 \\ 0 \leq x_i \leq 1 \end{array} \right\}.$$

Consider the constraint $x_3 + x_4 \leq 2$. This is a valid inequality for X , but it is also valid for P_1 . Hence, it would not improve the formulation P_1 .

Thus, to improve a formulation using valid inequalities, **a valid inequality must “cut off” some noninteger point in the formulation that is not within the convex hull of the integer solutions**. Even in these cases, identifying a valid inequality is often as hard as solving the original integer program.

In most cases, valid inequalities are constructed based upon some special structure of the integer program. For example, we can view the subtour elimination constraints of the traveling salesperson problem in Chapter 3 as valid inequalities for that problem, but these would not be valid for the region

P_1 in Example 14.7.

How do we find a valid inequality? In fact, we've already seen part of the idea when we first explored duality.

■ EXAMPLE 14.8

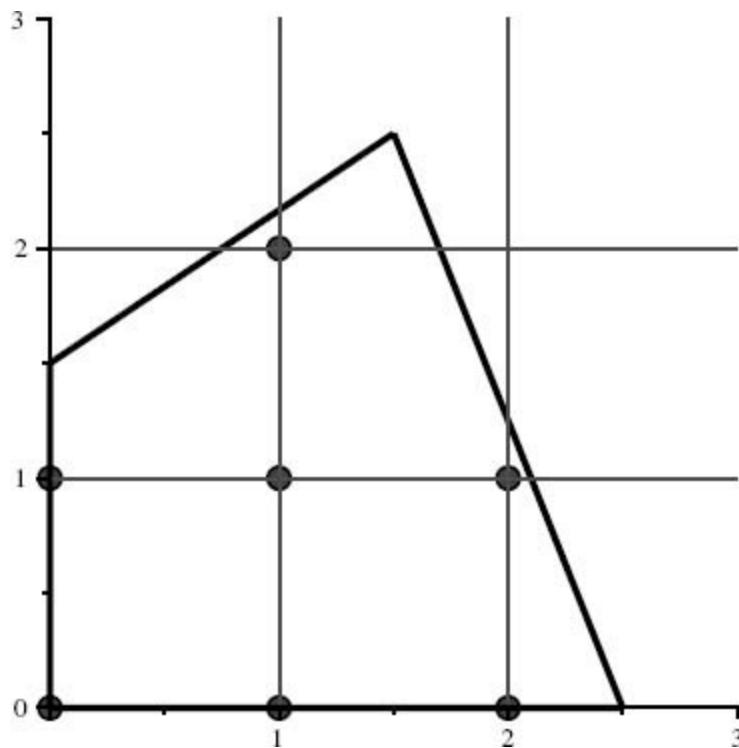
Consider the linear program

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & -4x + 6y \leq 9 \\ & 10x + 4y \leq 25 \\ (14.2) \quad & x, y \geq 0. \end{aligned}$$

The feasible region and all feasible integer solutions can be seen in [Figure 14.4](#). To generate a valid inequality for all feasible solutions (fractional and integer) to (14.2), suppose we multiply each of the constraints by a positive number and then add them. This was the approach with which we began our study of duality. For example, if we multiply the first constraint by $\frac{3}{38}$ and the second constraint by $\frac{5}{38}$, we get the inequality

$$\left(-4\left(\frac{3}{38}\right) + 10\left(\frac{5}{38}\right)\right)x + \left(6\left(\frac{3}{38}\right) + 4\left(\frac{5}{38}\right)\right)y \leq 9\left(\frac{3}{38}\right) + 25\left(\frac{5}{38}\right).$$

FIGURE 14.4 Feasible region for Example 14.8.



or $x + y \leq 4$. If we multiply the first constraint by $\frac{5}{38}$ and the second constraint by $\frac{1}{19}$, we get the valid inequality

$$y \leq \frac{5}{2}.$$

If we multiply the first constraint by 0.025 and the second constraint by 0.11, we get the valid inequality

$$x + 0.59y \leq 2.975.$$

If we consider only feasible solution where x and y have integer values, each of the constraints $x + y \leq 4$ and $y \leq \frac{5}{2}$ would have integer-valued left-hand sides since all coefficients are integers, but $x + 0.59y \leq 2.975$ would not. However, since $y \geq 0$, we have that $0.59y \geq 0$, so that $x \leq x + 0.059y$, which gives the valid inequality $x \leq 2.975$. Next, the constraints $y \leq \frac{5}{2}$ and $x \leq 2.975$ and $x \leq 2.975$ can be adjusted to $y \leq 2$ and $x \leq 2$ and still be valid for all integer feasible solutions; this is because the left-hand side of each constraint will have integer value, so the right-hand side can be rounded down. Thus, we have that the inequalities

$$x + y \leq 4$$

$$x \leq 2$$

$$y \leq 2$$

are all valid for every integer-valued feasible solution.

Let's consider what we did in this example. Since each constraint was a " \leq " constraint, if we multiply each constraint by a nonnegative number and add them together, any solution satisfying every constraint will also satisfy this new inequality. Formally, if we have the system of constraints

$$(14.3) \quad \sum_{j=1}^n a_{ij}x_j \leq b_i, \quad i \in \{1, \dots, m\}$$

and we multiply the i th constraint by $u_i \geq 0$ and sum these inequalities, we get the inequality

$$(14.4) \quad \sum_{i=1}^m \sum_{j=1}^n u_i a_{ij} x_j \leq \sum_{i=1}^m u_i b_i,$$

which is a valid inequality for (14.3). We can rewrite (14.4) by switching the order of summation on the left-hand side to get

$$(14.5) \quad \sum_{j=1}^n \left(\sum_{i=1}^m u_i a_{ij} \right) x_j \leq \sum_{i=1}^m u_i b_i.$$

Note that if A is the constraint matrix and $\mathbf{u} = (u_1, \dots, u_m)$, then

$$\sum_{j=1}^n \left(\sum_{i=1}^m u_i a_{ij} \right) x_j = \mathbf{u}^T A \mathbf{x}.$$

If each variable x_j is nonnegative, then, for any constant k , we have $\lfloor k \rfloor x \leq kx$, where again $\lfloor y \rfloor$ denotes the largest integer no greater than y . Thus, we can transform (14.5) into the valid inequality

$$(14.6) \quad \sum_{j=1}^n \left\lfloor \left(\sum_{i=1}^m u_i a_{ij} \right) \right\rfloor x_j \leq \sum_{i=1}^m u_i b_i.$$

In matrix–vector notation, if $\mathbf{u} \in \mathbb{R}^m$ and we let $\mathbf{v} = \lfloor \mathbf{u} \rfloor$ be the vector where each component $v_k = \lfloor u_k \rfloor$, then

$$\sum_{j=1}^n \left\lfloor \left(\sum_{i=1}^m u_i a_{ij} \right) \right\rfloor x_j = \lfloor \mathbf{u}^T A \rfloor \mathbf{x}.$$

Now suppose that each variable x_j is restricted to integer values. Since each coefficient on the left-hand side of (14.6) is now an integer, the left-hand-side value is an integer. Thus, we can reduce the right-hand side of (14.6) to $\lfloor \sum_{i=1}^m u_i b_i \rfloor$ and maintain its validity, generating the valid inequality

$$\sum_{j=1}^n \left\lfloor \left(\sum_{i=1}^m u_i a_{ij} \right) \right\rfloor x_j \leq \left\lfloor \sum_{i=1}^m u_i b_i \right\rfloor,$$

or

$$\lfloor \mathbf{u}^T A \rfloor \mathbf{x} \leq \lfloor \mathbf{u}^T \mathbf{b} \rfloor.$$

■ EXAMPLE 14.9

Using the data from Example 14.8, we have

$$A = \begin{bmatrix} -4 & 6 \\ 10 & 4 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 9 \\ 25 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}.$$

If we set $\mathbf{u} = \left(\frac{1}{5}, \frac{1}{4}\right)$, then

$$\begin{aligned} \lfloor \mathbf{u}^T A \rfloor &= \left\lfloor \begin{bmatrix} 1 & 1 \\ 5 & 4 \end{bmatrix} \begin{bmatrix} -4 & 6 \\ 10 & 4 \end{bmatrix} \right\rfloor \\ &= \left\lfloor \begin{bmatrix} \frac{17}{10} & \frac{11}{5} \end{bmatrix} \right\rfloor \\ &= \left[\left\lfloor \frac{17}{10} \right\rfloor \quad \left\lfloor \frac{11}{5} \right\rfloor \right] \\ &= [1 \quad 2] \end{aligned}$$

and

$$\begin{aligned} \lfloor \mathbf{u}^T \mathbf{b} \rfloor &= \left\lfloor \frac{9}{5} + \frac{25}{4} \right\rfloor \\ &= \left\lfloor \frac{161}{20} \right\rfloor \\ &= 8. \end{aligned}$$

This gives

$$x + 2y \leq 8,$$

which is valid for the convex hull of integer solutions to (14.2).

Summarizing, we get the following lemma.

Lemma 14.1 *Let $P = \{\mathbf{x} : A\mathbf{x} \leq \mathbf{b}; \mathbf{x} \geq \mathbf{0}\}$ and let $X = P \cap \mathbb{Z}^n$ be the set of solutions in P with all coordinates having integer value. If $\mathbf{u} \in \mathbb{R}^m$ satisfies $\mathbf{u} \geq \mathbf{0}$, then $\lfloor \mathbf{u}^T A \rfloor \mathbf{x} \leq \lfloor \mathbf{u}^T \mathbf{b} \rfloor$, or*

$$(14.7) \quad \sum_{j=1}^n \left\lfloor \left(\sum_{i=1}^m u_i a_{ij} \right) \right\rfloor x_j \leq \left\lfloor \sum_{i=1}^m u_i b_i \right\rfloor,$$

is a valid inequality for X .

Lemma 14.1 was first introduced by Gomory in 1958 [48] and then extended by Chvátal in 1973 [23]; the valid inequality (14.7) is often referred to as a *Chvátal–Gomory inequality (C–G inequality)*. A natural question related to

this lemma is whether every valid inequality is a C–G inequality.

Let $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ and let $X = P \cap \mathbb{Z}^n$ be the set of solutions in P with all coordinates having integer value. Then every valid inequality for X can be obtained by applying the rounding procedure given in Lemma 14.1 a finite number of times, where the current rounding procedure uses the original inequalities describing P and all C–G inequalities generated by previous iterations of the rounding procedure.

A proof can be found in Ref. [80]. This result is remarkable in the sense that only a finite number of roundings are needed, but this number can be exponentially large. It is also true that not all valid inequalities can be generated using just the original inequalities describing P .

■ EXAMPLE 14.10

Consider again the integer solutions to the feasible region of (14.2) given in [Figure 14.4](#). It should be clear that the inequality

$$x + y \leq 3$$

is valid for all integer solutions; however, this inequality cannot be formed by Lemma 14.1 using only the original inequalities. To see this, suppose u_1 and u_2 are the nonnegative multipliers for the constraints $-4x + 6y \leq 9$ and $10x + 4y \leq 25$, respectively. A C–G inequality would then be of the form

$$(-4u_1 + 10u_2)x + (6u_1 + 4u_2) \leq 9u_1 + 25u_2.$$

Consider the linear program

$$\begin{aligned} \min \quad & 9u_1 + 25u_2 \\ \text{s.t.} \quad & -4u_1 + 10u_2 \geq 1 \\ & 6u_1 + 4u_2 \geq 1 \\ & u_1, u_2 \geq 0. \end{aligned}$$

The constraints guarantee that each coefficient in the C–G inequality is at least 1 and the right-hand-side value is as small as possible. The optimal solution is $u_1 = \frac{3}{38}$, $u_2 = \frac{5}{38}$, with optimal value 4. Thus, we can generate the C–G inequality $x + y \leq 4$ from the constraints $-4x + 6y \leq 9$ and $10x + 4y \leq 25$ but not $x + y \leq 3$.

One question that is not addressed is which C–G inequalities are needed, at a minimum, to find an optimal integer solution by only solving the LP-relaxation. At present, one would need to generate all of them to guarantee that the optimal integer solution is found. We could find only a subset $Q\mathbf{x} \leq q$ of the valid inequalities, add those to our formulation, and solve the resulting problem using branch and bound. If the valid inequalities were chosen well, the relaxation gap would be small and thus branch and bound may solve the problem more efficiently. Unfortunately, for this approach to be effective we may have to add a very large number of inequalities, making the resulting linear programs harder (and more time-consuming) to solve.

When considering the addition of valid inequalities to strengthen the formulation of an integer program, we're typically interested only in strengthening the formulation near the integer optimal solution to our problem. If we were solving multiple problems over the same feasible region, but with different objectives, we would be more interested in a “global” formulation; since we're typically solving only one problem, more “local” formulation is useful.

■ EXAMPLE 14.11

Consider the linear program (14.2) from Example 14.8 whose feasible region and all feasible integer solutions can be seen in [Figure 14.4](#). The optimal solution to (14.2) is $(1.5, 2.5)$. If we add the valid inequality $x \leq 2$ (satisfied by all integer solutions), our optimal solution does not change. However, if we add the valid inequality $y \leq 2$, we get a different optimal solution of $(1.7, 2)$.

We could take an approach with valid inequalities similar to one we used in Chapter 3 to solve traveling salesperson problems. To solve TSPs, we started with an initial set of assignment constraints and generated only those subtour elimination constraints that were violated by the previous solution. In our current setting, we can solve our current formulation as a linear program and try to determine a valid inequality of the integer solutions that separates our current optimal solution from the convex hull of integer solutions. Such a valid inequality is called a *cutting plane* and this approach is called a **cutting plane algorithm**. It can be used with any valid inequality for our problem, not just C–G inequalities. Its general form is given in Algorithm 14.2.

You can probably see some inherent difficulties with a cutting plane algorithm. First, it is not always easy to find a cutting plane that separates our current solution from the convex hull of integer solutions. This **separation problem** of finding such a valid inequality can often be as difficult as solving the integer program itself. Second, the number of additional valid inequalities needed before an optimal integer solution is found can be very large. Thus, a simpler approach would be to stop a cutting plane algorithm after a fixed number of iterations (if an integer solution has not been found) and then solve the resulting formulation using branch and bound.

In the next two sections, we will focus on two cutting plane algorithms—one that uses C–G inequalities as cutting planes and the other for problems with only 0–1 variables.

Algorithm 14.2 General Cutting Plane Algorithm

Consider the integer program

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & \mathbf{x} \in X = P \cap \mathbb{Z}^n, \end{aligned}$$

where P is a formulation of X .

Step 0: Set $t = 0$ and $P^t = P$.

Step 1: Iteration t : Solve the linear program

$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & \mathbf{x} \in P^t. \end{aligned}$$

Let $\mathbf{x}^*(t)$ be the optimal solution.

Step 2: If $\mathbf{x}^*(t)$ is integer, stop. $\mathbf{x}^*(t)$ is the optimal integer solution to the integer program.

Step 3: Find a valid inequality $\mathbf{q}^T \mathbf{x} \leq q_0$ for X that is violated by $\mathbf{x}^*(t)$. Set $P^{t+1} = P^t \cap \{ \mathbf{x} : q^T \mathbf{x} \leq q_0 \}$, increase t by 1, and repeat Step 1.

14.4 GOMORY'S CUTTING

PLANE ALGORITHM

Suppose we have an integer program and a fractional solution \mathbf{x}^* to its LP-relaxation. Can we find a valid inequality of the integer solutions that is not satisfied by \mathbf{x}^* ? In other words, can we find a valid inequality that will “cut off” the current (fractional) optimal solution from the convex hull of integer solutions.

■ EXAMPLE 14.12

Let’s consider problem (14.2), but this time we’ll write it in canonical form

$$\begin{aligned} \max \quad & 3x + 2y \\ \text{s.t.} \quad & -4x + 6y + s_1 = 9 \\ & 10x + 4y + s_2 = 25 \\ (14.8) \quad & x, y, s_1, s_2 \geq 0. \end{aligned}$$

At the optimal solution $(1.5, 2.5, 0, 0)$, we have x, y as the basic variables and s_1, s_2 as the nonbasic variables. Solving the constraints for x and y using

$$\mathbf{x}_B = B^{-1}\mathbf{b} - B^{-1}N\mathbf{x}_N$$

gives

$$x = 1.5 + \frac{2}{38}s_1 - \frac{3}{38}s_2$$

$$y = 2.5 - \frac{5}{38}s_1 - \frac{2}{38}s_2.$$

Rewriting y ’s equation gives

$$y + \frac{5}{38}s_1 + \frac{2}{38}s_2 = 2.5.$$

If we employ the rounding principles we used to create the C–G inequalities and round down the coefficients of s_1 and s_2 , we generate the inequality $y \leq 2.5$. Since we want y to be integer-valued, we can round down the right-hand side and get the valid inequality $y \leq 2$. Note that this is one of the valid inequalities we generated earlier for this problem.

How did this happen? In Section 14.3, we needed to provide the constraint

multipliers \mathbf{u} before we could generate the valid inequality, but now it “magically” appeared. In fact, if you look closely, the multipliers are there.

■ EXAMPLE 14.13

If we consider the basic variables x and y for the linear program (14.8), the corresponding basis matrix B is

$$B = \begin{bmatrix} -4 & 6 \\ 10 & 4 \end{bmatrix}$$

and its inverse is

$$B^{-1} = \begin{bmatrix} -\frac{2}{38} & \frac{3}{38} \\ \frac{5}{38} & \frac{2}{38} \end{bmatrix}.$$

Note that the last row of B^{-1} contains the multipliers used in Example 14.8 to obtain the valid inequality $y \leq 2$.

In general, suppose we have the integer program

$$\max \quad \mathbf{c}^T \mathbf{x}$$

s.t.

$$A\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} \geq \mathbf{0}, \text{ integer}$$

where each entry of the matrix A and of the vector \mathbf{b} are integers. If we solve the LP-relaxation, we know that the matrix A can be partitioned into basic and nonbasic components B and N , respectively, and that the basic variables \mathbf{x}_B can be written in terms of the nonbasic variables

$$\mathbf{x}_B + B^{-1}N\mathbf{x}_N = B^{-1}\mathbf{b}.$$

Suppose basic variable x_{Bi} has fractional value in this solution. Its equation can be written in the form

$$x_{Bi} + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{b}_i,$$

where \bar{a}_{ij} is the (i, j) th entry of the matrix $B^{-1}N$ and \bar{b}_i is the i th component of the vector $B^{-1}\mathbf{b}$. Incorporating the rounding procedure used in Section 14.3, we get the valid inequality

$$x_{B_i} + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \bar{b}_i$$

by first rounding down the coefficients on the left-hand side, and then the valid inequality

$$(14.9) \quad x_{B_i} + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor$$

by rounding down the right-hand side value, since all variables are to have integer value. Note that our current optimal solution to the LP-relaxation violates (14.9). Thus, (14.9) is a cutting plane that we can add to form a stronger formulation to our problem. As noted in Example 14.13, (14.9) is in fact a C–G inequality whose constraint multipliers are the i th row of the matrix B^{-1} .

This cutting plane algorithm, which uses C–G inequalities, is known as the **Gomory Cutting Plane Algorithm** and is shown in Algorithm 14.3.

Algorithm 14.3 Gomory's Cutting Plane Algorithm

Consider the integer program

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & \mathbf{x} \in X = P \cap \mathbb{Z}^n, \end{aligned}$$

where P is a formulation of X .

Step 0: Set $t = 0$ and $P^t = P$.

Step 1: Iteration t : Solve the linear program

$$\begin{aligned} & \max \quad \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \\ & \mathbf{x} \in P^t. \end{aligned}$$

Let $\mathbf{x}^*(t)$ be the optimal solution.

Step 2: If $\mathbf{x}^*(t)$ is integer, stop. $\mathbf{x}^*(t)$ is the optimal integer solution to the integer program.

Step 3: Let $x_{B_i}^*$ have fractional value in the current solution. Generate the cutting plane

$$x_{B_i} + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j + s_{B_i} = \lfloor \bar{b}_i \rfloor$$

from the optimal basis (B, N) . Note that the slack variable s_{B_i} can be

constrained to be integer as well, since all variables and coefficients in the equation are integer valued.

Step 4: Set $P^{t+1} = P^t \cap \{x : x_{B_i} + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j + s_{B_i} = \lfloor \bar{b}_i \rfloor\}$, increase t by 1, and repeat Step 1.

■ EXAMPLE 14.14

Consider again the integer program (14.8). Solving its LP-relaxation gives the optimal solution (1.5, 2.5, 0, 0). In Example 14.12, we generated the cutting plane $y \leq 2$. Adding the slack variable s_3 and the constraint $y + s_3 = 2$ into our problem and resolving gives the optimal solution $(x, y, s_1, s_2, s_3) = (1.7, 2, 3.8, 0, 0)$. If we rewrite the constraints in terms of the basic variables, we get

$$\begin{aligned} x &+ \frac{1}{10}s_2 - \frac{2}{5}s_3 = \frac{17}{10} \\ y &+ s_3 = 2 \\ s_1 &+ \frac{2}{5}s_2 - \frac{38}{5}s_3 = \frac{19}{5}. \end{aligned}$$

Now consider the fractional variable x and its basis equation. This generates the valid inequality

$$x - s_3 \leq 1$$

or $x - s_3 + s_4 = 1$. Adding this equation to our formulation and resolving gives the optimal solution $(\frac{13}{6}, \frac{5}{6}, \frac{38}{3}, 0, \frac{7}{6}, 0)$ with basis equations

$$\begin{aligned} x &+ \frac{1}{6}s_2 - \frac{2}{3}s_4 = \frac{13}{6} \\ y &- \frac{1}{6}s_2 + \frac{5}{3}s_4 = \frac{5}{6} \\ s_1 &+ \frac{5}{3}s_2 - \frac{38}{3}s_4 = \frac{38}{3} \\ s_3 &+ \frac{1}{6}s_2 - \frac{5}{3}s_4 = \frac{7}{6}. \end{aligned}$$

Consider the equation

$$x + \frac{1}{6}s_2 - \frac{2}{3}s_4 = \frac{13}{6}.$$

This generates the cutting plane

$$x - s_4 \leq 2 \text{ or } x - s_4 + s_5 = 2.$$

Adding this equation to our formulation and resolving gives the optimal solution $(2, 1, 1, 1, 1, 0, 0)$. Since all the variables are integer, our optimal integer solution to the original problem is $(2, 1, 1, 1)$.

Algorithm 14.3 was first given by Gomory [48] and was proven to converge after a finite number of iterations in a subsequent study by Gomory [49], assuming that the cutting planes are properly chosen; see the book by Nemhauser and Wolsey [68] for more details. Unfortunately, for many years after the algorithm was presented, this approach was rarely (if ever) used to solve actual integer programs. In fact, many researchers figured that since early computational work had shown poor convergence results (the algorithm required extremely large numbers of iterations), the practical use of this approach was negligible. However, in the 1990s, a study was done along with additional research that found, if properly used, these cutting planes can be computationally effective. For more information, see the paper by Cornuéjols [24].

14.5 VALID INEQUALITIES FOR 0–1 KNAPSACK CONSTRAINTS

In most cases, valid inequalities are constructed based upon some special structure of the integer program. For the general case, very few valid inequalities are known. In fact, when valid inequalities are used for general integer programs, they typically come from those known for knapsack problems. Hence, we shall consider a class of valid inequalities for the 0–1 knapsack problem.

■ EXAMPLE 14.15

Suppose our relaxed feasible region is

$$P = \left\{ \mathbf{x} \in \mathbb{R}^4 : \begin{array}{l} 83x_1 + 57x_2 + 40x_3 + 19x_4 \leq 100 \\ 0 \leq x_i \leq 1 \end{array} \right\},$$

and that our current LP solution is $(1, \frac{11}{61}, 0, 0)$. In the integer case, we note that both x_1 and x_2 cannot have value 1 at the same time; this observation generates the valid inequality

$$x_1 + x_2 \leq 1.$$

Note that this valid inequality cuts off the solution $(1, \frac{11}{61}, 0, 0)$.

In general, if we have a constraint of the form

$$(14.10) \quad a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b,$$

where each $a_i > 0$ and each $x_i \in \{0, 1\}$, a valid inequality for the convex hull of integer solutions is

$$(14.11) \quad \sum_{i \in C} x_i \leq |C| - 1,$$

where C is a collection of $|C|$ indices such that

$$\sum_{i \in C} a_i > b.$$

Such a set C of indices is known as a **cover** of the knapsack constraint (14.10), and constraint (14.11) is known as a **cover inequality**. A cover C is a **minimal cover** if, for every $k \in C$, $C - \{k\}$ is not a cover.

■ EXAMPLE 14.16

Consider the constraint

$$25x_1 + 32x_2 + 40x_3 + 8x_4 + 15x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 \leq 100,$$

where each $x_k \in \{0, 1\}$. The set $C = \{2, 3, 4, 7\}$ is a cover since $32 + 40 + 8 + 37 > 100$, but it is not a minimal cover since $C - \{4\} = \{2, 3, 7\}$ is also a cover. The set $\{2, 3, 7\}$ is a minimal cover.

Given a constraint (14.10) for a 0–1 knapsack problem, if C is a cover of (14.10), then

$$\sum_{i \in C} x_i \leq |C| - 1$$

is a valid inequality for the feasible region

$$P = \{\mathbf{x} \in \{0, 1\}^n : a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b\}.$$

If C be a minimal cover of (14.10) and we let

$$E(C) = C \cup \{k : a_k \geq a_j, \quad j \in C\},$$

then

$$(14.12) \quad \sum_{i \in E(C)} x_i \leq |C| - 1$$

is also a valid inequality for P . The set $E(C)$ is called the **extended cover** of (14.10).

■ EXAMPLE 14.17

Consider again the constraint

$$25x_1 + 32x_2 + 40x_3 + 8x_4 + 15x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 \leq 100,$$

where each $x_k \in \{0, 1\}$. The set $C = \{1, 2, 6, 9\}$ is a minimal cover, so the constraint

$$x_1 + x_2 + x_6 + x_9 \leq 3$$

is a valid inequality for the constraint. The extended cover

$$E(C) = \{1, 2, 3, 6, 7, 9\} = C \cup \{3, 7\}$$

generates the valid inequality

$$x_1 + x_2 + x_3 + x_6 + x_7 + x_9 \leq 3.$$

How do we find such violated cover inequalities? Suppose, when solving a 0–1 knapsack problem, we obtain the fractional solution \mathbf{x}^* for some LP-relaxation. We want to know if any cover inequality (14.11) is violated by \mathbf{x}^* for some set C . One way is to let y_i denote whether index i will be in C , that is,

$$y_i = \begin{cases} 1, & \text{if } i \in C, \\ 0, & \text{otherwise.} \end{cases}$$

In this case, we want $\sum_{i \in C} a_i > b$,

$$\sum_{i \in C} a_i \geq b + 1,$$

since all coefficients a_i and b are integers. We also want

$$\sum_{i \in C} x_i^* > |C| - 1.$$

These two requirements can be formulated, using variables y_i , as

$$\begin{aligned} \sum_{i=1}^n a_i y_i &\geq b + 1 \\ \sum_{i=1}^n x_i^* y_i &> \sum_{i=1}^n y_i - 1 \iff 1 > \sum_{i=1}^n (1 - x_i^*) y_i. \end{aligned}$$

Since our goal is to find 0–1 values for y_i satisfying these requirements, we formulate the following knapsack problem

$$\begin{aligned} \xi^* = \min \quad & \sum_{i=1}^n (1 - x_i^*) y_i \\ \text{s.t.} \quad & \sum_{i=1}^n a_i y_i \geq b + 1 \\ & y_i \in \{0, 1\}, \end{aligned} \tag{14.13}$$

which is an equivalent form of our “traditional” knapsack problem. If the optimal value ξ^* is less than 1, then

$$C = \{j : y_j = 1\}$$

is a cover whose cover inequality (14.11) is violated by \mathbf{x}^* . Hence, to identify a violated cover inequality (14.11), we need to solve a knapsack problem. Note also that if we have identified a violated cover inequality, then the extended cover inequality (14.12) is also violated.

What if our problem is not a knapsack problem but has more than one constraint? Simple, we treat each individual constraint as a knapsack constraint and attempt to find a violated cover inequality for it. Since an inequality that is valid for one constraint of an integer program is valid for the entire problem, this allows us to identify multiple cutting planes to add to our formulation.

■ EXAMPLE 14.18

Let’s consider the problem

$$\begin{aligned}
\max \quad & 18x_1 + 43x_2 + 25x_3 + 38x_4 + 60x_5 + 15x_6 + 29x_7 + 50x_8 + 41x_9 \\
\text{s.t.} \quad & \\
& 25x_1 + 32x_2 + 40x_3 + 8x_4 + 15x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 \leq 100 \\
& 60x_1 + 34x_2 + 29x_3 + 57x_4 + 41x_5 + 36x_6 + 19x_7 + 47x_8 + 50x_9 \leq 100 \\
& x_k \in \{0, 1\}, \quad k = 1, \dots, 9.
\end{aligned}$$

Solving the LP-relaxation yields $\mathbf{x}^* = (0, 1, 0, 0, 1, 0, 1, 0.12766, 0)$ with value 138.383. If we consider the first constraint and search for a cover inequality violated by \mathbf{x}^* , we'd want to solve the integer program

$$\begin{aligned}
\xi^* = \min \quad & y_1 + 0y_2 + y_3 + y_4 + 0y_5 + y_6 + 0y_7 + 0.87234y_8 + y_9 \\
\text{s.t.} \quad & \\
& 25y_1 + 32y_2 + 40y_3 + 8y_4 + 15y_5 + 23y_6 + 37y_7 + 19y_8 + 27y_9 \leq 100 \\
& y_k \in \{0, 1\}, \quad k = 1, \dots, 9.
\end{aligned}$$

Its optimal solution is $(0, 1, 0, 0, 1, 0, 1, 1, 0)$ with value $\xi^* = 0.87234 < 1$, and hence the cover $C = \{2, 5, 7, 8\}$ generates a violated cover inequality. C is a minimal cover, and its extended cover $E(C) = \{2, 3, 5, 7, 8\}$ generates the cutting plane

$$x_2 + x_3 + x_5 + x_7 + x_8 \leq 3.$$

Doing the same for the second constraint yields the solution $(0, 1, 0, 0, 1, 0, 1, 1, 0)$ with value $\xi^* = 0.87234 < 1$, and hence the cover $C = \{2, 5, 7, 8\}$ generates a violated cover inequality. This time, though, C is not a minimal cover, but $C' = \{5, 7, 8\}$ is minimal. Its extended cover $E(C') = \{1, 4, 5, 7, 8, 9\}$ generates the cutting plane

$$x_1 + x_4 + x_5 + x_7 + x_8 + x_9 \leq 2.$$

After adding these two valid inequalities to the LP-relaxation and solving, we get the optimal solution $\mathbf{x}^* = (0, 1, 0, 0, 1, 0, 0.785714, 0.214286, 0)$ with value 136.5. Using this solution and the first constraint to generate (14.13), we get an optimal value of $\xi^* = 1$, and hence cannot identify a violated cover inequality. Using the second constraint, we find an optimal value of $\xi^* = 0.785724 < 1$ from cover $C = \{2, 5, 8\}$. C is a minimal cover and $E(C) = \{1, 2, 4, 5, 8, 9\}$ generates the cutting plane

$$x_1 + x_2 + x_4 + x_5 + x_8 + x_9 \leq 2.$$

Adding this constraint to our formulation gives an optimal solution of $\mathbf{x}^* = (0,$

$1, 0, 0, 1, 0.166667, 1, 0, 0$ with value 134.5. Solving (14.13) with the first constraint to identify a violated cover inequality, we find that the cover $C = \{2, 5, 6, 7\}$ generates such an inequality ($\xi^* = 0.833333$). With $E(C) = \{2, 3, 5, 6, 7\}$, we generate the cutting plane

$$x_2 + x_3 + x_5 + x_6 + x_7 \leq 3.$$

Doing the same for the second constraint yields an optimal $\xi^* = 0.833333$ from cover $C = \{2, 5, 6\}$. Its extended cover $E(C) = \{1, 2, 4, 5, 6, 8, 9\}$ generates the cutting plane

$$x_1 + x_2 + x_4 + x_5 + x_6 + x_8 + x_9 \leq 2.$$

After adding these two constraints to the formulation and solving, we get the optimal solution $\mathbf{x}^* = (0, 0.890909, 0.218182, 0, 1, 0, 0.890909, 0, 0.109091)$ with value 134.073. With this solution, we cannot identify a violated cover inequality for the first constraint by solving (14.13). However, the second constraint generates the cover $C = \{2, 3, 5\}$ ($\xi^* = 0.890909$), whose extended cover $E(C) = \{1, 2, 3, 4, 5, 8, 9\}$ generates the violated cover inequality

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_8 + x_9 \leq 2.$$

Adding this constraint to our formulation and solving yields the optimal solution $\mathbf{x}^* = (0, 1, 0, 0, 1, 0, 1, 0, 0)$ with value 132. Since it is integral, this is the optimal integer solution.

Unlike the problem given in Example 14.18 where we were able to obtain an optimal integer solution using only cuts obtained from violated cover inequalities, most cutting plane approaches do not find an optimal integer solution before it ends. This is for a variety of reasons: other cutting planes are required in order to find an integer solution but are not explored, or, as in the case of Gomory cuts, the number of iterations/cuts required to find an integer solution may be too large, and hence an algorithm may place a limit on the number of cuts obtained before quitting.

14.6 BRANCH-AND-CUT ALGORITHMS

Suppose we have finished generating cutting planes and an integer solution

has not been found. What can be done? The revised problem (including the obtained cutting planes) is now solved using branch and bound. Typically, the number of nodes required to solve the revised problem is smaller than that of the original problem because the integrality gap at the root node is often smaller for the revised problem.

■ EXAMPLE 14.19

Let's consider the problem

$$\max \quad 50x_1 + 53x_2 + 50x_3 + 40x_4 + 45x_5 + 55x_6 + 44x_7 + 50x_8 + 51x_9$$

s.t.

$$25x_1 + 32x_2 + 40x_3 + 20x_4 + 23x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 \leq 105$$

$$40x_1 + 34x_2 + 29x_3 + 30x_4 + 20x_5 + 36x_6 + 25x_7 + 47x_8 + 40x_9 \leq 105$$

$$x_k \in \{0, 1\}, \quad k = 1, \dots, 9.$$

Solving this problem using branch and bound (without adding cutting planes as in Section 14.5) solves the problem in 57 nodes, with an initial LP-relaxation value of 181.242. If we try to identify violated cover inequalities as we did in the similar problem from Example 14.18, we get the improved formulation

$$(14.14) \max \quad 50x_1 + 53x_2 + 50x_3 + 40x_4 + 45x_5 + 55x_6 + 44x_7 + 50x_8 + 51x_9$$

s.t.

$$25x_1 + 32x_2 + 40x_3 + 20x_4 + 23x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 \leq 105$$

$$40x_1 + 34x_2 + 29x_3 + 30x_4 + 20x_5 + 36x_6 + 25x_7 + 47x_8 + 40x_9 \leq 105$$

$$x_1 + x_2 + x_3 + x_5 + x_6 + x_8 + x_9 \leq 3$$

$$x_2 + x_3 + x_5 + x_6 + x_7 \leq 3$$

$$x_1 + x_2 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3$$

$$x_1 + x_3 + x_4 + x_5 + x_6 + x_8 + x_9 \leq 3$$

$$x_3 + x_4 + x_5 + x_7 \leq 3$$

$$x_1 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3$$

$$x_k \in \{0, 1\}, \quad k = 1, \dots, 9.$$

Solving the LP-relaxation gives a value of 177.15 but does not yield an integer solution, so branch and bound is needed, requiring 31 nodes. For each formulation, the optimal value to the integer program is 158.

A natural thought at this point would be to consider adding valid inequalities at each node of the branch-and-bound tree in order to improve those formulations. If branch and bound requires a large amount of time and nodes, then doing more work to improve the formulation at each node seems as a viable way to reduce the overall time required to solve the problem. However, there is a small problem. If a valid inequality is generated at some node in the tree, it probably is valid only at subsequent nodes down the tree and not necessarily at every node in the tree.

■ EXAMPLE 14.20

Suppose, in Example 14.19, we are at a node of the branch-and-bound tree where we have fixed $x_3 = 1$. If we consider the first constraint, then at this node it looks like

$$\begin{aligned} 25x_1 + 32x_2 + 20x_4 + 23x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 &\leq 105 - 40 \\ &= 65. \end{aligned}$$

For the subproblem at this node, the inequality

$$x_2 + x_5 + x_6 \leq 2$$

is valid but clearly is not valid for the original problem (and hence for every node in the tree) since $\mathbf{x} = (0, 1, 0, 0, 1, 1, 0, 0, 0)$ is feasible.

We are left with two options. First, we can generate valid inequalities for a given node and then only use them at this node and successors to this node in the tree. This would require keeping track of which nodes an inequality is valid for, which can be difficult if we search the tree using a method other than depth-first search. A second approach would be to transform an inequality that is valid at one node into one that is valid for every node. To illustrate, let's continue with the preceding example.

■ EXAMPLE 14.21

In Example 14.20, we generated the valid inequality

$$x_2 + x_5 + x_6 \leq 2$$

for the constraint (where $x_3 = 1$)

$$\begin{aligned} 25x_1 + 32x_2 + 20x_4 + 23x_5 + 23x_6 + 37x_7 + 19x_8 + 27x_9 &\leq 105 - 40 \\ &= 65. \end{aligned}$$

We need to modify this inequality to also be valid when $x_3 = 0$. It seems reasonable that such a modification would require the coefficient of x_3 to increase from 0 and that the right-hand side must also increase from 2, and that both parts increase by the same amount α , making the new inequality of the form

$$\alpha x_3 + x_2 + x_5 + x_6 \leq 2 + \alpha.$$

Note that, if $x_3 = 1$, we get the original inequality, and that if $x_3 = 0$, we get an inequality with an adjusted right-hand side.

So how can we find α ? We only need to consider the case when $x_3 = 0$. We would want the maximum of $x_2 + x_5 + x_6$ to be no larger than $2 + \alpha$ for every feasible solution where $x_3 = 0$. This leads to the problem

$$\begin{aligned} \zeta^* &= \max && x_2 + x_5 + x_6 \\ \text{s.t.} & && \\ & 32x_2 + 23x_5 + 23x_6 \leq 65 + 40 = 105 \\ & x_2, x_5, x_6 \in \{0, 1\}. \end{aligned}$$

Note that since the other variables do not contribute to the objective function, we can assume that their value is 0 in this problem. Since $\zeta^* = 3$, we would want $\alpha \geq \zeta^* - 2 = 1$. Setting $\alpha = 1$ yields

$$x_3 + x_2 + x_5 + x_6 \leq 3,$$

which is valid for the original problem.

This concept of increasing coefficients and right-hand sides to extend a valid inequality of a subset of feasible solutions to all feasible solutions is often referred to as **lifting**. We have seen in Example 14.21 the basic idea of lifting the coefficient of a variable whose value is fixed to 1. A similar idea can be derived for cases when the variable is fixed to 0. These two approaches are given in the following lemmas.

Lemma 14.2 Suppose $S \subseteq \{0, 1\}^n$ and

$$\sum_{j=2}^n \beta_j x_j \leq \beta_0$$

is a valid inequality for $S^1 = S \cap \{\mathbf{x} : x_1 = 1\}$. If $S^0 = S \cap \{\mathbf{x} : x_1 = 0\} = \emptyset$, then $x_1 \geq 1$ is a valid inequality for S . If $S^0 = \emptyset$, then

$$\alpha x_1 + \sum_{j=2}^n \beta_j x_j \leq \beta_0 + \alpha$$

is a valid inequality for S for any $\alpha \geq \zeta^* - \beta_0$, where

$$\zeta^* = \max \quad \sum_{j=2}^n \beta_j x_j$$

s.t.

$$\mathbf{x} \in S^0.$$

Lemma 14.3 Suppose $S \subseteq \{0, 1\}^n$ and

$$\sum_{j=2}^n \beta_j x_j \leq \beta_0$$

is a valid inequality for $S^0 = S \cap \{\mathbf{x} : x_1 = 0\}$. If $S^1 = S \cap \{\mathbf{x} : x_1 = 1\} = \emptyset$, then $x_1 \leq 0$ is a valid inequality for S . If $S^1 = \emptyset$, then

$$\alpha x_1 + \sum_{j=2}^n \beta_j x_j \leq \beta_0$$

is a valid inequality for S for any $\alpha \leq \beta_0 - \zeta^*$, where

$$\zeta^* = \max \quad \sum_{j=2}^n \beta_j x_j$$

s.t.

$$\mathbf{x} \in S^1.$$

Since the values of α in both Lemmas 14.2 and 14.3 are bounded, if we choose α to satisfy the inequalities at equality, we are said to do **maximum lifting**.

■ EXAMPLE 14.22

Suppose we have

$$S = \left\{ \mathbf{x} \in \{0, 1\}^6 : 8x_1 + 7x_2 + 9x_3 + 5x_4 + 14x_5 + 12x_6 \leq 30 \right\}$$

and consider

$$\begin{aligned} S' &= S \cap \{x \in \{0, 1\}^6 : x_3 = x_5 = 0, x_6 = 1\} \\ &= \{(x_1, x_2, x_4) \in \{0, 1\}^3 : 8x_1 + 7x_2 + 5x_4 \leq 18\}. \end{aligned}$$

A valid inequality for S' is $x_1 + x_2 + x_4 \leq 2$. In the following calculations, all the liftings done will be maximum liftings. Suppose we first lift variable x_3 . Since $x_3 = 0$, we use Lemma 14.3 and solve

$$\begin{aligned} \zeta_3^* &= \max \quad x_1 + x_2 + x_4 \\ \text{s.t.} \\ 8x_1 + 7x_2 + 5x_4 &\leq 18 - 9 = 9 \\ x_1, x_2, x_4 &\in \{0, 1\}. \end{aligned}$$

Since $\zeta_3^* = 1$, we have $\alpha_3 = 2 - 1 = 1$, which generates the valid inequality $x_1 + x_2 + x_3 + x_4 \leq 2$ for

$$S'' = \{(x_1, x_2, x_3, x_4) \in \{0, 1\}^4 : 8x_1 + 7x_2 + 9x_3 + 5x_4 \leq 18\}.$$

Now let's lift x_6 . Since $x_6 = 1$, we use Lemma 14.2 and solve

$$\begin{aligned} \zeta_6^* &= \max \quad x_1 + x_2 + x_3 + x_4 \\ \text{s.t.} \\ 8x_1 + 7x_2 + 9x_3 + 5x_4 &\leq 18 + 12 = 30 \\ x_1, x_2, x_3, x_4 &\in \{0, 1\}. \end{aligned}$$

We have $\zeta_6^* = 4$, which gives $\alpha_6 = 4 - 2 = 2$, yielding the valid inequality $x_1 + x_2 + x_3 + x_4 + 2x_6 \leq 4$ for

$$S''' = \{(x_1, x_2, x_3, x_4, x_6) \in \{0, 1\}^5 : 8x_1 + 7x_2 + 9x_3 + 5x_4 + 12x_6 \leq 30\}.$$

Finally, lifting x_5 gives the problem

$$\begin{aligned} \zeta_5^* &= \max \quad x_1 + x_2 + x_3 + x_4 + 2x_6 \\ \text{s.t.} \\ 8x_1 + 7x_2 + 9x_3 + 5x_4 + 12x_6 &\leq 30 - 14 = 16 \\ x_1, x_2, x_3, x_4, x_6 &\in \{0, 1\}. \end{aligned}$$

We have $\zeta_5^* = 2$, which gives $\alpha_5 = 4 - 2 = 2$. Thus, we have generated the valid inequality

$$x_1 + x_2 + x_3 + x_4 + 2x_5 + 2x_6 \leq 4$$

for S . Note that this is not a cover inequality.

It is important to note that the order in which we lift the variables can alter the coefficients, leading to different valid inequalities.

■ EXAMPLE 14.23

Consider the set of solutions

$$S = \left\{ \mathbf{x} \in \{0, 1\}^5 : 13x_1 + 10x_2 + 9x_3 + 8x_4 + 6x_5 \leq 20 \right\}$$

and consider

$$S' = S \cap \{\mathbf{x} : x_1 = x_5 = 0\}.$$

A valid inequality for S' is $x_2 + x_3 + x_4 \leq 2$. If we first lift x_1 , we find that the maximum lifting coefficient $\alpha_1 = 2$. Next, lifting x_5 gives a maximum lifting coefficient of $\alpha_5 = 0$. If, however, we first lift x_5 and then x_1 , we get maximum lifting coefficients of $\alpha_1 = \alpha_5 = 1$.

Using Lemmas 14.2 and 14.3, we can generate additional valid inequalities that may be violated by our current relaxation solution but could not have been identified previously. Thus, we have more opportunities to generate cutting planes than previously, as the next example illustrates.

■ EXAMPLE 14.24

Suppose we continue from the formulation (14.14) for the integer program given in Example 14.19. If we solve the LP-relaxation to this problem, we get the solution

$$\mathbf{x} = (0, 0.213889, 0.584259, 0.798148, 1, 0.617593, 0.584259, 0,$$

If we attempt to find a violated cover inequality by solving the integer program (14.13) for the second constraint, we find that the best cover is $C = \{3, 4, 5, 6\}$, but that this does not generate a violated cover inequality. Normally, we would stop our cutting plane generation and proceed to solve this problem via branch and bound; however, suppose we decide to lift the remaining variables into our cover inequality $x_3 + x_4 + x_5 + x_6 \leq 3$ to see if we can generate a violated inequality. If we lift x_2 using Lemma 14.3 by initially assuming $x_2 = 0$, we find that its lifted coefficient is 1, which generates the lifted inequality

$$x_2 + x_3 + x_4 + x_5 + x_6 \leq 3,$$

which is violated by our current solution. If we proceed by first solving (14.13) to find an initial minimal cover and then lift the remaining variables using Lemma 14.3, we generate an additional nine valid inequalities, which results in a relaxation whose value is 174.772. Solving this problem using branch and bound now requires only 11 nodes.

An added benefit of lifting coefficients for valid inequalities is that we can generate valid inequalities at one node of the branch-and-bound tree and use them at any other node. This approach of combining branch-and-bound with cutting planes is referred to as **branch-and-cut** and is the method of choice of modern solvers for integer programs. An outline of branch-and-cut is given in Algorithm 14.4.

Algorithm 14.4 Branch-and-Cut Algorithm

Input: Integer program $P = \{\max \mathbf{c}^T \mathbf{x} : \mathbf{x} \in P \cap \mathbb{Z}^n\}$

Initialization Let R be list of unsolved subproblems. Initially, set $R = \{P\}$. Let v_{best} be value of best-known feasible solution to P (if none exists, set $v_{\text{best}} = -\infty$).

(3) while $R = \emptyset$ **do**

Choose and remove problem P' from R .

repeat

Solve LP-relaxation of P' to obtain solution \mathbf{x} .

if P' is infeasible **then**

go to (3).

else if \mathbf{x} is integer-valued **then**

Compare value of \mathbf{x} to v_{best} and update v_{best} if necessary.

go to (3).

else

Look for violated inequalities and add them to LP-relaxation .

end if

until No violated inequalities are identified.

Partition P' into subproblems and add them to R .

end while

14.7 COMPUTATIONAL ISSUES

We've mentioned a few times in this chapter that due to the lack of an optimality condition for general integer programs, solving these problems can be time-consuming. Typically (and this is especially true for classes of integer programs such as those examined in Chapter 4), a heuristic is used to first generate an initial incumbent solution and branch-and-bound is then used to either verify that it is optimal or produce a better solution. When we have obtained the optimal solution, though, we often spend large amounts of time verifying this fact. This is the computational drawback associated with integer programs.

Because of this difficulty, much work has occurred over the last few decades on efficiency options for branch-and-bound, cutting plane, and branch-and-cut algorithms. Researchers have proposed rules and options geared toward speeding up these algorithms for many problems, knowing that their approach will not be efficient for all instances. Modern software packages have incorporated many such improvements and provide the user with the ability to turn these options on or off as they desire. In this section, we will comment on some of these so that you are familiar with the available options.

Branch-and-Cut Options

Because branch-and-cut algorithms are just branch-and-bound methods with the possible generation of cutting planes at each node, we will first examine those options common to both. As noted in Section 14.2, the main choices in a branch-and-cut (-bound) algorithm are (1) which subproblem or node to consider next and (2) how to partition our current problem into other subproblems via branching. The goal of each option is to decrease the amount of time and computations required to obtain the optimal integer solution.

Node Selection We mentioned in Section 14.2 two common node selection approaches: best-first and depth-first. Best-first focuses on lowering the global upper bound by partitioning those subproblems with larger relaxation values and (hopefully) finding an integer solution with larger objective value, while depth-first focuses on finding feasible integer solutions by successively decomposing subsets of the original problem. Each has its own benefits and

detractions. When an integer solution is obtained by a best-first search, it is generally of large value, however, such solutions typically are not found until many subproblems have been generated, which creates large computer storage issues. Depth-first search finds feasible solutions more quickly, but their objective value is typically not as high. They require little storage for subproblems, since each successive problem is similar to the previous one examined. Another approach focuses on providing an estimate of the best integer solution value at each subproblem. Such *best estimate* methods are often not the default setting and require more calculations than the others.

Most software packages use a hybrid approach, utilizing the strengths of both the best-first and depth-first methods. Initially, a depth-first approach is used to obtain feasible integer solutions; later, a best-first approach is used to improve the upper bound, which tries to verify optimality. In some cases, such as the optimization package CPLEX, the user can choose the “emphasis” of the search (feasibility, optimality, balanced), which indicates how the hybrid method should work. Packages often allow the user to choose which approach to use, but default to the hybrid method.

Branching Branching indicates how the current subproblem is to be decomposed. In our examples earlier in this chapter, we simply chose one of the variables x_k whose value was fractional, that is,

$$x_k = v_k + f_k,$$

where $v_k \in \mathbb{Z}$ and $0 < f_k < 1$. We then decomposed the problem by adding the constraints $x_k \leq v_k$ and $x_k \geq v_k + 1$ to form two new subproblems. If there are multiple fractional variables, which one should be chosen?

One simple approach is to choose the variable that is “most fractional,” that is, the variable that maximizes $\min\{f_k, 1 - f_k\}$; variables that have fractions close to 0.5 are more likely to be chosen. However, just because a variable is “very fractional” does not automatically make it a good choice for branching variable.

Branching variables greatly influence the value of the upper bound generated by the problem relaxation; thus, a good branching decision should attempt to predict which variable will lower the upper bound the most. One way to do this associates with each integer-restricted variable x_j two estimates P_j^+ and P_j^- and P_j^- and P_j^+ for the per-unit decrease in the objective value if we fix x_j

to either $v_j + 1$ or v_j , respectively. These values are often called the *up-* and *down-pseudocosts*. The estimated change in value can be calculated by $D_j^+ = P_j^+(1 - f_j)$ and $D_j^- = P_j^- f_j$. One simple way to compute these pseudocosts is to use the values of both the current relaxation value z_{LP} and the relaxation values z_{LP}^+ and z_{LP}^- obtained by fixing x_j to v_j and $v_j + 1$ as follows:

$$P_j^+ = \frac{z_{\text{LP}}^+ - z_{\text{LP}}}{1 - f_j} \quad \text{and} \quad P_j^- = \frac{z_{\text{LP}} - z_{\text{LP}}^-}{f_j}.$$

Once the pseudocosts have been calculated for each potential branching variable, we can choose our branching variable by selecting either the variable that maximizes $D_j^+ + D_j^-$, the total changes, or the variable that maximizes the minimum of D_j^+, D_j^- . Each option is typically available in a software package.

Another approach that has proven quite useful is to estimate the change in objective function by solving the new linear program partially, by doing only a (small) fixed number of iterations, typically using dual simplex. The dual objective values are then compared, with the variable generating the smallest change is used. This approach is known as *strong branching* and has been successfully used in large integer programs. However, it is computationally expensive, since many linear programs need to be solved at each node before a branching variable is selected.

Heuristics Another option that has proven important is the running of heuristic methods at a node to attempt to generate a feasible integer solution. Many optimization packages employ their own general heuristic methods to produce integer solutions, and there are many specialized algorithms in use for specific classes of problems, such as the TSP or network design problems. The user is often given the option not only of using heuristics, but also how often. Again, there is a trade-off between the time required to run the heuristic and the potential time savings generated by finding good integer solutions.

A recent general heuristic that many packages are including is the *feasibility pump* introduced by Fischetti et al. [37]. In this heuristic, the LP-relaxation is first solved, which generates a feasible solution \mathbf{x} . Those variables that have fractional value are rounded (up or down) to their nearest integer value, generating a (not necessarily) feasible solution $\tilde{\mathbf{x}}$. We then

solve a linear program that tries to find a feasible solution “close to” \tilde{x} . This process repeats until we identify a feasible integer solution or until a time limit is reached. Computational tests indicate that the feasibility pump generates good initial integer solutions in a reasonable amount of time.

Cutting Planes

Whether done on their own or as part of a branch-and-cut algorithm, cutting plane methods have many aspects that can be tuned to improve their computational performance. These include determining which classes of valid inequalities to search for, how thorough this search should be, and how many to generate.

We previously mentioned two general types of valid inequalities: C–G inequalities that are valid for any integer program and cover inequalities for 0–1 programs. In addition, we mentioned subtour elimination constraints for the traveling salesperson problem (Chapter 3) and vehicle routing problems (Chapter 4). There are other general classes of valid inequalities often explored in optimization packages, including mixed integer rounding (MIR) cuts (Marchand and Wolsey [63]), flow cover inequalities (Padberg, Van Roy, and Wolsey [69]), and lift-and-project inequalities (Balas, Ceria, and Cornuéjols, [5]). Some of these are specialized only for 0–1 programs or mixed-integer programs, but many can be used for any problem.

Another issue that becomes prevalent is how aggressively we search for violated inequalities. For example, when we searched for violated cover inequalities in Section 14.5, we searched for the most violated inequality by solving a knapsack problem. Since this problem itself may be difficult, various heuristics are often employed to solve this problem. In addition, we saw that even when a violated cover inequality cannot be identified, we may lift some of the remaining variables to obtain a violation. Parameters may be set by the user to indicate how much work we are willing to do each time to identify such a cut. In addition, since it is possible to generate a very large number of cutting planes, every algorithm stops generating new ones based on some criteria. These include a fixed number of cuts allowed or if the addition of new cuts does not change the objective value “enough.”

Lifting is another issue for cutting planes. We have already seen that the order variables are lifted can influence their value. Since there is no best

order to lift the variables, researchers have done experiments on determining which order to suggest. Gu, Nemhauser, and Savelsbergh [52] conducted a study on generating and lifting cover inequalities, where they specified a particular algorithm involving the generation of a potentially violated cover inequality and a lifting order that worked well on some standard test cases.

One final issue is the storage of the cutting planes generated during an algorithm. If we were to keep all these inequalities every time we solve an LP-relaxation, our resulting linear program would be too large and require too much time to solve, even using the dual simplex method. Many packages use a *cut pool* to maintain these generated inequalities. At each iteration, the cuts are examined and added to the model if our current relaxed solution violates it or removed from the model if it is satisfied by the current solution and has been satisfied for a fixed number of iteration. This helps maintain a reasonable sized problem and keeps a readily available pool of constraints to check without the generation of additional ones.

Summary

In this chapter, we explored various approaches to solving integer programs, including cutting plane methods and branch-and-bound (-cut). We also discussed some computational issues that can arise with each of these methods. Since integer programs are generally tough to solve, much research has been done in generating methods (both heuristic and exact) that reduce the computational time required to solve a problem using these algorithms. It is hopefully clear that minor adjustments to these approaches can lead to vast changes in efficiency, and finding the proper set of “tweaks” is an ongoing and useful process.

EXERCISES

14.1 Solve the integer program

$$\begin{aligned}
 \max \quad & 7x + 3y \\
 \text{s.t.} \quad & \\
 & 2x + 5y \leq 28 \\
 & 8x + 3y \leq 45 \\
 (14.15) \quad & x, y \geq 0, \text{ integer}
 \end{aligned}$$

using complete enumeration. Plot the feasible region first to help identify all integer solutions.

14.2 Solve the integer program (14.15) from Exercise 14.1 using branch-and-bound in which the next subproblem is chosen using a depth-first approach. Indicate the order you solve the subproblems. You may use an optimization package to solve the subproblems.

14.3 Solve the integer program (14.15) from Exercise 14.1 using branch-and-bound where the next subproblem chosen is based upon a best-first approach. Indicate the order you solve the subproblems. You may use an optimization package to solve the subproblems.

14.4 Solve the integer program

$$\begin{aligned}
 \max \quad & 5x + 2y \\
 \text{s.t.} \quad & \\
 & 12x - 7y \leq 84 \\
 & 6x + 10y \leq 69 \\
 & x, y \geq 0, \text{ integer}
 \end{aligned}$$

using branch-and-bound with a best-first subproblem selection rule. Indicate the order you solve the subproblems. You may use an optimization package to solve the subproblems.

14.5 Solve the integer program

$$\begin{aligned}
\min \quad & 5x_1 + 4x_2 + 3x_3 \\
\text{s.t.} \quad & \\
& x_1 + 3x_2 + 4x_3 \leq 11 \\
& 2x_1 + 2x_2 + x_3 \leq 7 \\
& 3x_1 + 4x_2 + 2x_3 \leq 12 \\
& x_1, x_2 \geq 0, \text{ integer} \\
& x_3 \geq 0
\end{aligned}$$

using a branch-and-bound approach with a best-first rule to choose the next subproblem. Note that not every variable is integer restricted.

14.6 Solve the knapsack problem

$$\begin{aligned}
\max \quad & 20x_1 + 16x_2 + 25x_3 + 14x_4 + 9x_5 \\
\text{s.t.} \quad & \\
& 3x_1 + 2x_2 + 5x_3 + 4x_4 + 2x_5 \leq 13 \\
& x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}
\end{aligned}$$

using branch-and-bound applying a best-first approach to choosing the next subproblem. Indicate the order you solve the subproblems. Use the results of Exercise 9.17 to solve the LP-relaxations for each subproblem.

14.7 Repeat Exercise 14.6 using a depth-first approach to select the next subproblem.

14.8 Repeat Exercises 14.6 and 14.7, but first use the greedy heuristic (Algorithm 5.1) prior to starting the branch-and-bound method. How many fewer nodes are explored since you have an initial feasible solution?

14.9 Solve the knapsack problem in Exercise 14.6 using complete enumeration.

14.10 Repeat Exercises 14.6–14.8 for the following problem:

$$\begin{aligned}
\max \quad & 18x_1 + 43x_2 + 25x_3 + 38x_4 + 60x_5 + 15x_6 + 29x_7 + 50x_8 + 41x_9 \\
\text{s.t.} \quad & \\
& 60x_1 + 34x_2 + 29x_3 + 57x_4 + 41x_5 + 36x_6 + 19x_7 + 47x_8 + 50x_9 \leq 100 \\
& x_k \in \{0, 1\}, \quad k = 1, \dots, 9.
\end{aligned}$$

14.11 Given the integer program (14.15) determine whether or not the following are valid inequalities.

(a) $x + y \leq 8$
(c) $x + 2y \leq 12$

(b) $5x + 4y \leq 36$
(d) $3x + y \leq 16$

14.12 Given the polytope

$$P = \left\{ \mathbf{x} : \begin{array}{l} -4x + 6y \leq 9 \\ 10x + 4y \leq 25 \\ x, y \geq 0 \end{array} \right\},$$

find a C-G valid inequality of $X = P \cap \mathbb{Z}^2$ that is violated by the extreme point $(2.5, 0)$.

14.13 Given the polytope

$$P = \left\{ \mathbf{x} : \begin{array}{l} 3x_1 - 4x_2 + 5x_3 \leq 9 \\ 3x_1 + 2x_2 + x_3 \leq 7 \\ x_1, x_2, x_3 \geq 0 \end{array} \right\},$$

find a C-G valid inequality of $X = P \cap \mathbb{Z}^3$ that is violated by the extreme point $(\frac{13}{6}, 0, \frac{1}{2})$.

14.14 Solve the integer program

$$\max \quad 17x + 12y$$

s.t.

$$10x + 7y \leq 40$$

$$x + y \leq 5$$

$$x, y \geq 0, \text{ integer}$$

using Gomory's cutting plane algorithm. Use your optimization package to solve each linear program.

14.15 Solve the integer program

$$\max \quad 3x + 5y$$

s.t.

$$x - y \geq -1$$

$$3x + 2y \leq 3$$

$$6x + y \leq 14$$

$$x, y \geq 0, \text{ integer}$$

using Gomory's cutting plane algorithm. Use your optimization package to solve each linear program.

14.16 Solve the integer program

$$\begin{aligned}
\min \quad & 3x - y \\
\text{s.t.} \quad & 2x + 5y \leq 30 \\
& 8x + 3y \leq 48 \\
& 2x - y \leq 7 \\
& -x + 2y \leq 7 \\
& x, y \geq 0, \text{ integer}
\end{aligned}$$

using Gomory's cutting plane algorithm. Use your optimization package to solve each linear program.

14.17 Suppose our current binary integer program (each variable $x_k \in \{0, 1\}$) contains the constraint

$$4x_1 + 6x_2 + 2x_3 + 5x_4 + 7x_5 \leq 12.$$

Identify all minimum covers and state their corresponding cover inequalities.

14.18 Suppose our current binary integer program (each variable $x_k \in \{0, 1\}$) contains the constraint

$$7x_1 + 8x_2 + 6x_3 + 10x_4 + 9x_5 + 11x_6 \leq 38.$$

Identify all minimum covers and state their corresponding cover inequalities.

14.19 Identify the extended covers for each of the minimum covers found in Exercise 14.17.

14.20 Identify the extended covers for each of the minimum covers found in Exercise 14.18.

14.21 Let

$$X = \left\{ \mathbf{x} \in \{0, 1\}^6 : 9x_1 + 8x_2 + 7x_3 + 6x_4 + 3x_5 + 2x_6 \leq 20 \right\}.$$

Find a cover inequality of X violated by each of the following solutions.

(a) $\mathbf{x} = (0.5, 1, 0.5, 0, 1, 0.5)$.

(b) $\mathbf{x} = (1, \frac{1}{3}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, 1)$.

14.22 Let

$$X = \left\{ \mathbf{x} \in \{0, 1\}^7 : 15x_1 + 8x_2 + 12x_3 + 10x_4 + 9x_5 + 11x_6 + 14x_7 \leq 42 \right\}.$$

Find a cover inequality of X violated by each of the following solutions.

(a) $\mathbf{x} = (0.4, 0.75, 0.75, 0.8, 0.2, 0, 0.75)$.

(b) $\mathbf{x} = (0.8, 0.125, 0, 0.2, 1, 1, 0.5)$.

14.23 Suppose we have

$$S = \left\{ \mathbf{x} \in \{0, 1\}^6 : 9x_1 + 8x_2 + 7x_3 + 6x_4 + 5x_5 + 4x_6 \leq 17 \right\}$$

and consider

$$\begin{aligned} S' &= S \cap \{\mathbf{x} \in \{0, 1\}^6 : x_1 = x_3 = x_6 = 0\} \\ &= \left\{ (x_2, x_4, x_5) \in \{0, 1\}^3 : 8x_2 + 6x_4 + 5x_5 \leq 17 \right\}. \end{aligned}$$

The cover inequality $x_2 + x_4 + x_5 \leq 2$ is a valid inequality for S' . Lift this inequality to generate another valid inequality of S . Use maximum liftings at each step and lift the variables in the order x_1, x_3, x_6 .

14.24 Suppose we have

$$S = \left\{ \mathbf{x} \in \{0, 1\}^7 : 15x_1 + 10x_2 + 12x_3 + 11x_4 + 9x_5 + 11x_6 + 13x_7 \leq 40 \right\}$$

and consider

$$\begin{aligned} S' &= S \cap \{\mathbf{x} \in \{0, 1\}^7 : x_1 = x_3 = x_4 = 0\} \\ &= \left\{ (x_2, x_5, x_6, x_7) \in \{0, 1\}^4 : 10x_2 + 9x_5 + 11x_6 + 13x_7 \leq 40 \right\}. \end{aligned}$$

The cover inequality $x_2 + x_5 + x_6 + x_7 \leq 3$ is valid for S' . Using maximum liftings, lift this inequality in each of the following variable orders.

(a) x_3, x_4, x_1

(b) x_4, x_3, x_1

14.25 Suppose we have

$$S = \left\{ \mathbf{x} \in \{0, 1\}^6 : 9x_1 + 8x_2 + 7x_3 + 6x_4 + 5x_5 + 4x_6 \leq 17 \right\}$$

and consider

$$\begin{aligned} S' &= S \cap \{\mathbf{x} \in \{0, 1\}^6 : x_1 = x_2 = 0, x_5 = x_6 = 1\} \\ &= \left\{ (x_3, x_4) \in \{0, 1\}^2 : 7x_3 + 6x_4 \leq 9 \right\}. \end{aligned}$$

The cover inequality $x_3 + x_4 \leq 1$ is a valid inequality for S' . Lift this

inequality to generate another valid inequality of S under each of the following variable orders. Use maximum liftings at each step.

(a) x_5, x_6, x_1, x_2

(b) x_6, x_2, x_5, x_1

CHAPTER 15

SOLVING INTEGER PROGRAMS: MODERN HEURISTIC TECHNIQUES

In Chapter 14, we explored various approaches to solving integer programs exactly. Much of this dealt with ways to either decompose the problem into smaller feasible regions (branch and bound) or to reformulate our problem by adding additional constraints (cutting planes). Each approach attempts to reduce the computational time required to solve integer programs, which can take very long to solve; recall the computational effort needed to solve large traveling salesperson problems discussed in Chapter 3.

Another approach to solving integer programs is to use heuristic methods, which we first discussed in Chapter 5. These methods attempt to find a good solution in an efficient manner, but are not guaranteed to always find the optimal solution. However, some of the methods discussed in Chapter 5 have potential drawbacks that limit their effectiveness.

In this chapter, we explore a few modern heuristic approaches to discrete optimization problems. We highlight their strengths and discuss weaknesses for each method. Since there is no one “best” approach to a problem, it is useful to have as many techniques as possible available to solve a given problem.

15.1 REVIEW OF LOCAL SEARCH METHODS: PROS AND

CONS

Before we discuss new heuristic methods, it is useful to review the notion of local search from Chapter 5, which forms the backbone for many of the methods to be discussed. Local search algorithms begin with a complete feasible solution \mathbf{x} and search its neighborhood $N(\mathbf{x})$, that is, those solutions that are “close” to \mathbf{x} , to see if there is a solution \mathbf{y} with better objective value. If such a \mathbf{y} exists, we select this solution and repeat the process. This method is outlined in Algorithm 15.1.

Algorithm 15.1 General Local Search Algorithm

```
x ← generateInitialSolution().  
repeat  
    x ← FindBetterSolution(N(x))  
until no better solution in N(x) available
```

There are some issues with local search methods that need to be addressed. First, its performance (in terms of finding a near-optimal solution) greatly depends not only on the neighborhood $N(\mathbf{x})$ used for each feasible solution \mathbf{x} but also on how the next solution is chosen from the neighborhood, as well as the initial solution itself. In Chapter 5, we already saw that different neighborhood structures can be derived for the same feasible solution, so this choice is an important one. It needs to strike a balance between containing many solutions, which has a better chance of obtaining the optimal solution but requires a great deal of time to search, and containing too few solutions, which can be scanned very quickly but leaves little improvement options.

Choosing an improving solution from the neighborhood of the current solution is also vital to the performance of a local search method. Some algorithms use a greedy approach, where the best solution in the neighborhood is chosen. Others choose the first improving solution found, while some choose randomly from all improving solutions found. These selection methods are similar to those used to select entering variables in the simplex method, and as in the simplex method, various approaches have been suggested with none proving to be the best in all cases. Greedy methods not only have the advantage of quicker improvements but also can occasionally stop after fewer iterations. Random selection expands the number of possible

paths to local optimal solutions our algorithm could explore, but to truly benefit from randomization multiple runs of the method must be done.

In addition, local search methods stop once a local optimal solution is found, that is, one that has no improving solution within its neighborhood. This means that it is possible for an algorithm to search only a small part of the feasible region and ignore the remainder. We could run our local search approach multiple times, each time starting at a different solution (chosen by some randomization approach); however, as before we should do this many times to truly see the effect of randomization. If there are very few local optimal solutions, so that many initial solutions “funnel” to the same ones, then such multistart approaches would notice this, and reporting the best local optima found would give confidence to it being (possibly) the best solution; however, if there are many local optima, then a very large number of initial solutions would be needed to feel as confident about the quality of our final solution.

Because of these drawbacks, different strategies to guide the search process have been proposed. Known as **metaheuristics**, these methods typically incorporate some random component and often use search memory to guide future selections. They often attempt to avoid being “trapped” at a local optimal solution by using methods to “back out” of such solutions. In addition, their goal is often to balance the notion of *diversification*, or the exploration of much of the feasible region, and *intensification*, or the focusing around a small area for the best solutions. In the next few sections, we will introduce some of the more common metaheuristics: simulated annealing, tabu search, genetic algorithms, and GRASP.

15.2 SIMULATED ANNEALING

Simulated Annealing is probably one of the oldest metaheuristic approaches; its basic approach dates back to the study of Metropolis et al. [64] in 1953 describing an algorithm used to simulate the heating and cooling of solid material in a heat bath (annealing). In annealing, a solid is heated past its melting point, where the atoms become free of their initial positions and change randomly through various states. As the solid is cooled the atoms stick together, often resulting in structures that have lower internal energy.

Different cooling rates can produce different properties of the solid.

To turn this approach into a heuristic method for optimization problems, we emulate the cooling mechanism by a temperature T . Suppose, we are at a solution \mathbf{x} to our problem whose objective function is f and we randomly select a potential solution \mathbf{x}' . If $f(\mathbf{x}')$ is better than $f(\mathbf{x})$, we accept \mathbf{x}' as our current solution and continue. However, if it is not better, we will still accept \mathbf{x}' with probability $e^{-|f(\mathbf{x})-f(\mathbf{x}')|/T}$. This probability follows the *Boltzmann distribution*. Note that, for a fixed value T , the larger the difference between the function values of \mathbf{x} and \mathbf{x}' , the smaller the probability of acceptance of \mathbf{x}' . A general simulated annealing algorithm is given in Algorithm 15.2. In this formulation, some possible termination conditions are (1) when we reach a maximum number of iterations, (2) the temperature T gets close to 0 and (3) the current solution does not change after too many iterations; the most common condition is based upon the temperature.

Algorithm 15.2 General Simulated Annealing Algorithm (maximization problems)

```

 $\mathbf{x} \leftarrow \text{generateInitialSolution}()$ .
repeat
     $\mathbf{x}' \leftarrow \text{RandomSolution}(N(\mathbf{x}))$ 
    if  $f(\mathbf{x}')$  is better than  $f(\mathbf{x})$  then
         $\mathbf{x} \leftarrow \mathbf{x}'$ 
    else
         $p \leftarrow \text{generateRandomNumber}()$ 
        if  $p \leq e^{-\frac{|f(\mathbf{x})-f(\mathbf{x}')|}{T}}$  then
             $\mathbf{x} \leftarrow \mathbf{x}'$ 
        end if
    end if
     $T \leftarrow \text{UpdateTemp}(T)$ 
until termination conditions satisfied

```

Typically, the temperature cools according to a function $Q(T, k)$ of both the current temperature T and the iteration number k . This allows the temperature to cool at different rates during the algorithm's run, enabling the algorithm to balance the demands of diversification and intensification. For example, early in the search process T might be large and change at either a constant or a linear rate so as to examine different areas of the feasible region. Later in the process, we want T to decrease rapidly in order to converge to one solution. Other approaches where the temperature both increases and decreases have been suggested as a way to oscillate between the two demands. In fact, under

some general conditions on the cooling rate of T , the algorithm will converge to a global optimal solution with probability 1 (note that this does not necessarily mean the solution is the global optimal solution, but that such instances are extremely rare). However, such required cooling rates are not practical since they decrease the temperature too slowly.

When implementing a simulated annealing algorithm for a problem, there are three initial decisions we need to make: (1) initial temperature, (2) cooling method, and (3) stopping condition. Each of these affects the performance of the method, and so some thought must be put into each decision. For example, the initial temperature T_0 should be chosen high enough so that the acceptance rate for “worse” solutions is high in the initial iterations. The choice of an appropriate T_0 can be problem specific since knowing the magnitude of solutions in the neighborhood of the initial solutions should influence our choice. Many implementations typically try various initial values before settling on a specific value.

The cooling schedule has probably the most influence on the performance of the algorithm. Its role is to both allow initial random fluctuations in the solutions early in the process and then prevent good solutions to be replaced by the worse ones later. Some implementations allow multiple solutions to be generated at each temperature and larger temperature shifts, while others generate only one solution per temperature, but then reduce the temperature more slowly. In practice, two common cooling schemes are a geometric approach

$$T_{k+1} = \alpha T_k,$$

where α is close to 1, and the function

$$T_{k+1} = \frac{T_k}{1 + \beta T_k},$$

where β is close to 0. Some implementations have included a temperature increase using

$$T_{k+1} = \frac{T_k}{1 - \gamma T_k}$$

when a worsening move is rejected, although this is not standard.

■ EXAMPLE 15.1

Consider the following 15-city traveling salesperson problem whose distance matrix is

$$(15.1) \quad \begin{bmatrix} - & 48 & 42 & 34 & 45 & 32 & 25 & 21 & 46 & 31 & 23 & 25 & 24 & 46 & 36 \\ 48 & - & 27 & 24 & 45 & 46 & 21 & 37 & 39 & 43 & 30 & 38 & 44 & 45 & 48 \\ 42 & 27 & - & 47 & 48 & 33 & 37 & 46 & 34 & 27 & 29 & 33 & 31 & 28 & 32 \\ 34 & 24 & 47 & - & 46 & 35 & 20 & 25 & 32 & 33 & 26 & 47 & 26 & 22 & 33 \\ 45 & 45 & 48 & 46 & - & 48 & 34 & 34 & 38 & 40 & 31 & 38 & 50 & 27 & 45 \\ 32 & 46 & 33 & 35 & 48 & - & 22 & 36 & 36 & 22 & 24 & 24 & 29 & 45 & 29 \\ 25 & 21 & 27 & 20 & 34 & 22 & - & 46 & 21 & 27 & 39 & 34 & 21 & 28 & 40 \\ 21 & 37 & 46 & 25 & 34 & 36 & 46 & - & 28 & 26 & 23 & 31 & 32 & 38 & 29 \\ 46 & 39 & 34 & 32 & 38 & 36 & 21 & 28 & - & 34 & 28 & 45 & 31 & 48 & 30 \\ 31 & 43 & 27 & 33 & 40 & 22 & 27 & 26 & 34 & - & 39 & 44 & 42 & 27 & 28 \\ 23 & 30 & 29 & 26 & 31 & 24 & 39 & 23 & 28 & 39 & - & 36 & 45 & 37 & 30 \\ 25 & 38 & 33 & 47 & 38 & 24 & 34 & 31 & 45 & 44 & 36 & - & 39 & 42 & 32 \\ 24 & 44 & 31 & 26 & 50 & 29 & 21 & 32 & 31 & 42 & 45 & 39 & - & 27 & 36 \\ 46 & 45 & 28 & 22 & 27 & 45 & 28 & 38 & 48 & 27 & 37 & 42 & 27 & - & 25 \\ 36 & 48 & 32 & 33 & 45 & 29 & 40 & 29 & 30 & 28 & 30 & 32 & 36 & 25 & - \end{bmatrix}.$$

Suppose that our initial tour is $\mathbf{x} = 1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow 15 \rightarrow 1$, which has distance 548. We will set our initial temperature at $T_0 = 500$ and use $\alpha = 0.99$.

In the first iteration, we randomly swap the cities in locations 6 and 12, resulting in the tour

$$\begin{aligned} \mathbf{x}' &= 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 12 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \\ &\rightarrow 6 \rightarrow 13 \rightarrow 14 \rightarrow 15, \end{aligned}$$

which has distance 528. Since this is a better distance, we keep this solution. We now update our temperature to $T_1 = \alpha T_0 = (0.99)500 = 495$. In the next iteration, we propose to swap the 1st and 10th positions, resulting in the solution

$$\begin{aligned} \mathbf{x}' &= 10 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 12 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 1 \rightarrow 11 \\ &\rightarrow 6 \rightarrow 13 \rightarrow 14 \rightarrow 15, \end{aligned}$$

which has distance 511; we keep this solution as well, and our new temperature is $T_2 = (0.99)495 = 490.05$. We next propose to swap the 10th and 15th positions, which leads to the solution

$$\begin{aligned} \mathbf{x}' &= 10 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 12 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 15 \rightarrow 11 \\ &\rightarrow 6 \rightarrow 13 \rightarrow 14 \rightarrow 1, \end{aligned}$$

which has distance 526. Since this has a higher distance than our current

solution, we generate a random number and compare it to

$$e^{(511-526)/T_k} = e^{(-15)/490.05} = 0.969855.$$

Suppose, our randomly generated number is 0.0.755436, which implies that our proposed solution is kept. We then update our temperature T_k and proceed.

When we continued this for 1000 iterations, we found the tour

$$\begin{aligned} \mathbf{x} = & 13 \rightarrow 9 \rightarrow 15 \rightarrow 3 \rightarrow 10 \rightarrow 16 \rightarrow 12 \rightarrow 1 \rightarrow 8 \rightarrow 11 \\ & \rightarrow 5 \rightarrow 14 \rightarrow 4 \rightarrow 2 \rightarrow 7, \end{aligned}$$

which has distance 381. [Figure 15.1](#) shows a graph of the solution values generated over these 1000 iterations. Notice how the solution value varies greatly for the first 400 or so iterations, but then the temperature T_k becomes small enough to reduce the chance a worse solution is accepted. This illustrates the typical behavior of a simulated annealing approach. In [Figure 15.2](#), we have the results of a second run of 1000 iterations, but this time with $\alpha = 0.95$. Note that in this case our final solution is only of length 383, and that we seem to converge in much fewer iterations.

FIGURE 15.1 Solution values from simulated annealing run.

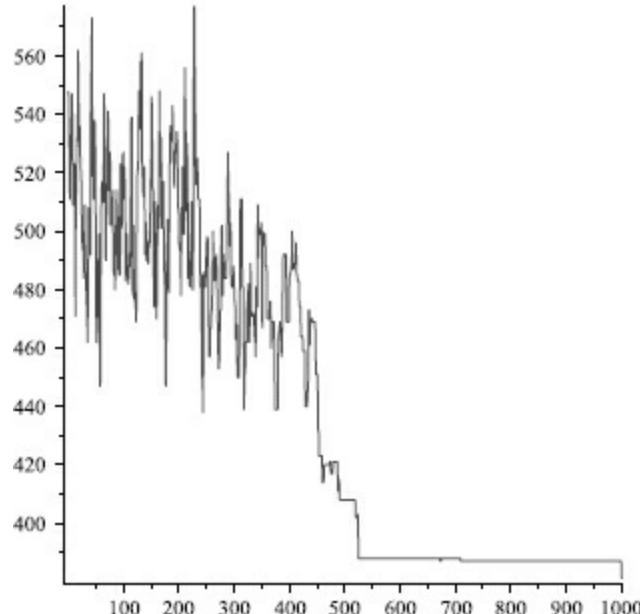
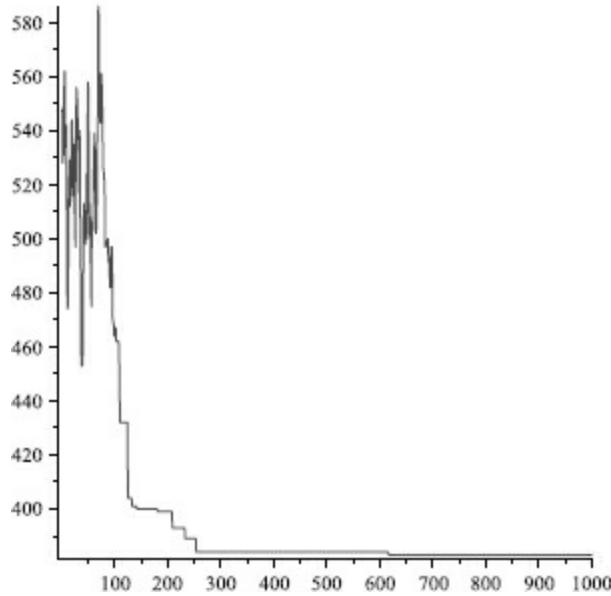


FIGURE 15.2 Solution values from 2nd simulated annealing run.



Simulated annealing has the benefit of being a very simple method to implement, requiring few parameter choices for the user. Its performance has historically been good, but typically other metaheuristic methods have produced better results. In fact, it is often used as part of other metaheuristics. For more information on simulated annealing, see Henderson, et al. [57].

15.3 TABU SEARCH

Simulated annealing moves out of local optimal solutions through a probabilistic approach; *Tabu Search* uses deterministic means (through the use of memory) to do the same. In tabu search, information from previous iterations is used to guide current and future moves. The ideas behind tabu search were first given by Glover [42] and by Hansen [54].

Tabu search works similar to a local improvement approach until it reaches a local optima, at which point it uses short-term memory to move away from this local optima. Recent solutions (or the moves that generated them) are kept in a *tabu list* and are thus ineligible for consideration for a set amount of time. This both restricts the neighborhood of available solutions and helps prevent cycling to recently visited solutions. Elements are kept in the tabu list according to a *tabu tenure*, which dictates both the size of the list and how long of a memory to use. At each iteration, the best nontabu solution from the neighborhood of the current solution is chosen, and this process continues

until some termination condition is met, which is typically a maximum number of iterations.

■ EXAMPLE 15.2

To illustrate a tabu search method, we develop one for a TSP, using the specific 15-city data given in (15.1) from Example 15.1. We represent a tour by the sequence of cities visited, and the neighborhood of a given tour T is all tours T_k obtained from T by swapping the order of two cities. For example, given our 15-city TSP and the tour

$$\begin{aligned}T = 1 &\rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \\&\rightarrow 12 \rightarrow 13 \rightarrow 14 \rightarrow 15,\end{aligned}$$

one of the neighbors of T would be obtained by swapping the 4th and 12th cities, resulting in the tour

$$\begin{aligned}1 &\rightarrow 2 \rightarrow 3 \rightarrow 12 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \\&\rightarrow 4 \rightarrow 13 \rightarrow 14 \rightarrow 15.\end{aligned}$$

Given this definition of a neighborhood, one possible tabu list would simply be a collection of k recent tours, where k is some user-defined parameter indicating the size of the tabu list (and also the tabu tenure for each tour on the tabu list).

Keeping track of recent solutions in a tabu list can be impractical due to the size of the solutions. Instead, *solution attributes* are typically kept. In this case, the small adjustments made to move from one solution to another are kept in the tabu list, and it is these attributes that are made tabu for a period of time. Unfortunately, this can lead to problems since an attribute can be shared by multiple solutions; hence, instead of making tabu only one solution, many can be made tabu at that same time, including the potential global optimal solution. To remedy this, an *aspiration criteria* can be defined that allows tabu moves to be used. For example, we can set as an aspiration criteria that a tabu move can be used if it generates a better solution than our current best. An outline of generic tabu search approach is given in Algorithm 15.3.

■ EXAMPLE 15.3

For the TSP, instead of using tours as elements of our tabu list, one possible tabu attribute could be to prevent a specific element from being altered for

some tenure length. If we use the tour T and its neighbor T_1 from Example 15.2, since the 1st and 10th positions were swapped, we could make swapping the 1st element tabu for some fixed number of iterations. Note that this prevents multiple solutions from being used. For example, if we consider the tour

$$\begin{aligned} 1 &\rightarrow 2 \rightarrow 3 \rightarrow 12 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \\ &\rightarrow 4 \rightarrow 13 \rightarrow 14 \rightarrow 15 \end{aligned}$$

Algorithm 15.3 Tabu Search: General Form

```

x ← FindInitialSolution()
TabuList ← ∅
repeat
     $\hat{x}$  ← BestMove( $N(x)$ )
    if  $\hat{x} \notin$  TabuList then
         $x \leftarrow \hat{x}$ 
    else if  $\hat{x} \in$  TabuList and Aspiration Criteria met then
         $x \leftarrow \hat{x}$ 
    else
        Do not update x
    end if
    UpdateTabuList(x)
until Maximum Iterations Reached

```

and make altering the 4th position tabu, this would prevent the tour

$$\begin{aligned} 1 &\rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 5 \rightarrow 6 \rightarrow 12 \rightarrow 8 \rightarrow 9 \rightarrow 1 \rightarrow 11 \\ &\rightarrow 4 \rightarrow 13 \rightarrow 14 \rightarrow 15, \end{aligned}$$

obtained by swapping positions 4 and 7, from being considered, even though its length of 490 is better than the first tour's length of 503. To ensure that we can also improve upon the best-found solution (if such a tour is in our neighborhood), we could use as an aspiration criteria that we always choose a tabu move if it results in a tour that is better than our current best tour.

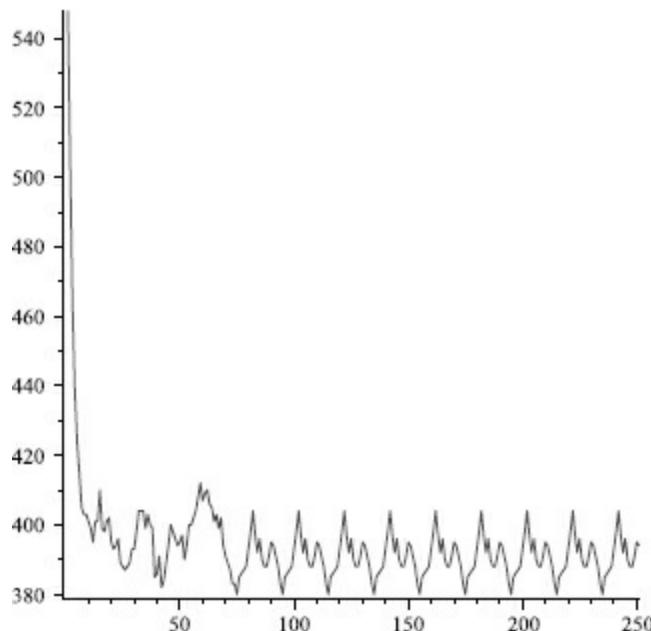
The search of the feasible region is dictated by the tabu tenure. A short tenure typically concentrates the search in a small area of the feasible region since moves can be repeated after only a few iterations. A long tenure supports diversification because it often forces the algorithm to explore larger (and different) regions. Tabu tenures need not be static, but can be changed throughout the process to reflect the needs of the search. If various solutions seem to be repeating themselves, the tenure may be increased to generate more diversification, while it may be lowered (to provide intensification) if there is not enough improvement made over an extended number of

iterations. This allows us to incorporate long-term memory into the search process.

■ EXAMPLE 15.4

Suppose, we run a tabu search approach on our 15-city TSP using the tabu list and aspiration criteria described in Example 15.2. If we run it for 250 iterations, using a tabu tenure of 3 iterations, we find the best tour of distance 380 after about 75 iterations. [Figure 15.3](#) shows the tour distances found over these 250 iterations. Note that the graph oscillates after finding this best tour, which indicates that our tenure is too small and we are not escaping far enough from this local optima. If we increase our tenure to 4, again get the tour of length 380, but the search is able to explore other areas of the feasible region; see [Figure 15.4](#) for a graph of the tour distances found. If we increase the tenure to 5, we find a tour of length 378 after about 50 iterations (see [Figure 15.5](#)).

[FIGURE 15.3](#) Solution values from tabu search (tenure = 3).



Tabu search is perhaps the most commonly used metaheuristic approach for operations research problems. Its flexibility allows it to be combined with other techniques in order to generate higher quality solutions. More information on tabu search can be found in the book by Glover and Laguna [44].

15.4 GENETIC ALGORITHMS

Each of the other metaheuristic techniques discussed in this chapter works with only one solution at a time. However, the class of algorithms known as *Genetic Algorithms* deals with sets of solutions, all at the same time. Genetic algorithms work by mimicking the biological process of evolution through the selection of parents, the producing of offspring, and the random mutation of traits in their creation. Such approaches were first described by Holland [60] and have been used to generate solutions for many difficult problems, primarily because they can quickly search large feasible regions.

FIGURE 15.4 Solution values from tabu search (tenure = 4).

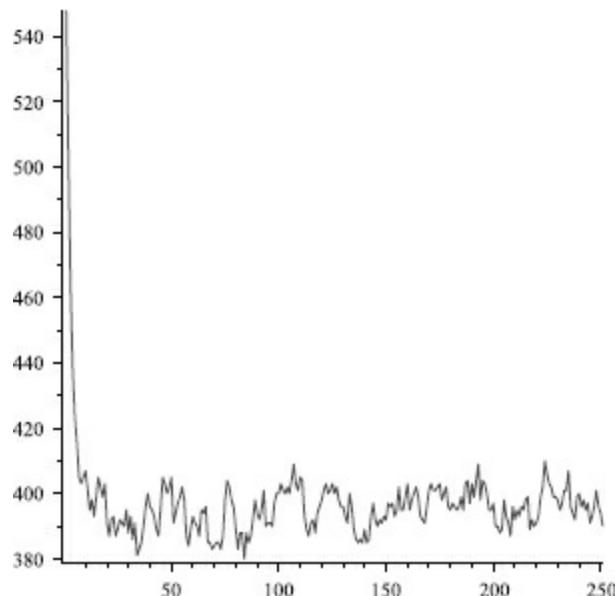
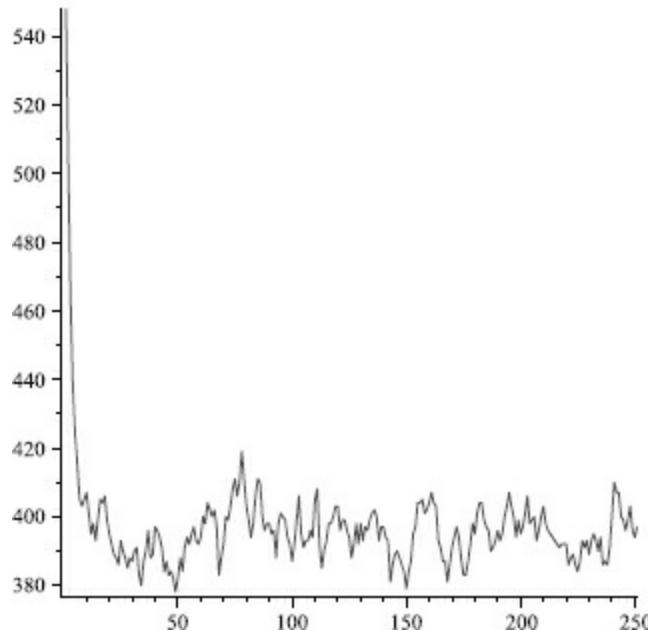


FIGURE 15.5 Solution values from tabu search (tenure = 5).



The basics of a genetic algorithm are easy to describe. Genetic algorithms begin by randomly generating an initial set, or *population*, of n solutions. Each solution is evaluated according to a *fitness function* that may or may not be related to the objective. From this population, a subset of parent solutions is selected; this is done either randomly or by incorporating the fitness function. Parents are then combined using a *crossover* operation to create offspring that will form the next generation of solutions. These offspring may experience some random changes or *mutations* that introduce diversity into the population. This process is repeated until some stopping criteria is met.

In genetic algorithms, the representation of a single solution is key. It is important that each solution is represented in such a way that enables the operations of evaluation, reproduction, and mutation. Typically, solutions are encoded either as a binary string or as a permutation of integers. Such encoded strings, often referred to as *chromosomes*, must be finite in length and easily manipulated. The initial population is generally constructed by randomly generating n chromosomes. Typical values of n are 50 or 100, which provides a reasonable amount of diversity while still being manageable. However, as with parameters in any algorithm, this value can be altered based upon experimental results.

■ EXAMPLE 15.5

Consider a set covering problem

$$\begin{aligned} \min \quad & \sum_{i=1}^n x_i \\ \text{s.t.} \quad & \sum_{i \in S_j} x_i \geq 1, \quad j \in \{1, \dots, m\} \\ & x_i \in \{0, 1\}, \end{aligned}$$

where each $S_j \subset \{1, \dots, n\}$. A natural encoding of the solutions is as a string of 0's and 1's, where each element corresponds to the value of a particular variable x_i . Simply using the objective function as a string's fitness function does not seem reasonable for this problem, since an infeasible solution could have lower objective value than a feasible one. A possible alternative fitness function could be

$$\text{Fitness}(\mathbf{x}) = \sum_{i=1}^n x_i + M(\text{number of violated constraints})$$

for some parameter M . For large enough values of M , infeasible solutions would have “worse” fitness than some feasible ones, but if we select M to be small enough, those infeasible solutions that are “nearly feasible” may not be penalized too much.

For a TSP, such a binary encoding is impractical due to its size (an n -city TSP requires a string whose length is roughly n^2). A more natural encoding is a string whose elements come from the set $\{1, 2, \dots, n\}$ and where each element is unique; such a string represents the order in which the cities are visited in a tour. For such an encoding, the length of a tour is a reasonable fitness, since feasibility is already considered in the encoding.

Each member of the population is evaluated for fitness, and this often determines whether it is a possible parent for the next generation. Typically, the better the fitness, the greater the chance it is part of the mating population. One example for selecting a set of possible parents is to first rank all n members of a population according to their fitness value, then assigning the probability that the k th best member is part of the mating population to be

$$P(k\text{th best chosen}) = \frac{2k}{n(n+1)}.$$

It is conceivable that a member is selected multiple times, where copies of it are made for the mating population. Another method is to simply take some percentage (say the top 75%) of the ranked solutions as the mating population. Also, some have suggested that the top members automatically belong in the mating population, although this is not standard. In fact, it is sometimes useful to allow “bad” solutions into a population since otherwise the population may become too homogeneous and similar to a local optimal solution.

Once the mating population has been determined, the reproduction phase begins. This entails the creation of a method to generate offspring using information from both parent’s chromosomes. Such *crossover* approaches randomly select a part of each parent’s chromosome at which it is to be “separated.” The first part of one parent is combined with the second part of the other parent. Depending upon the encoding, crossover techniques need to ensure that a reasonable offspring is generated. A selection scheme is also needed to determine the next generation of solutions. This can be done by randomly selecting^{1,2} n pairs from the mating population and performing a crossover with each pair to produce two new solutions. Some methods allow these selected parents to remain in the population (at a fixed probability) rather than perform a crossover.

■ EXAMPLE 15.6

Suppose that the encoding for our solutions is a binary string of length n . One possible crossover approach is to randomly select a position c and break each parent at this position. The offspring solutions are created by combining the first part (prebreak) of one parent with the second part of the other parent. For example, if we have the parents $(0, 1, 1, 0, 1, 0, 0, 1)$ and $(1, 0, 1, 1, 0, 1, 0, 1)$ and we randomly select position $c = 3$ as our break point, the first parent breaks into the substrings $(0, 1, 1)$ and $(0, 1, 0, 0, 1)$ while the second parent breaks into $(1, 0, 1)$ and $(1, 0, 1, 0, 1)$. These can then recombine into the solutions $(0, 1, 1, 1, 0, 1, 0, 1)$ and $(1, 0, 1, 0, 1, 0, 0, 1)$.

For encodings that are permutations, such as the one suggested for the TSP, more care has to be taken, since the encoding itself preserves feasibility. For example, if we had the parents

$$P_1 = (4, 3, 6, 5, 1, 2)$$

$$P_2 = (3, 5, 2, 1, 6, 4)$$

and used the above crossover technique after breaking each at position 3, we would get the offspring

$$O_1 = (4, 3, 6, 1, 6, 4)$$

$$O_2 = (3, 5, 2, 5, 1, 2),$$

which are clearly not feasible tours. One possible crossover technique is to first break the parents at some point c as before, copying the first c elements to an offspring, and then add the elements from the other parent in order without copying any element. If we use this approach on parents P_1 and P_2 above, and use the break point $c = 2$, we would have the offspring

$$O_1 = (4, 3, 5, 2, 1, 6)$$

$$O_2 = (3, 5, 4, 6, 1, 2).$$

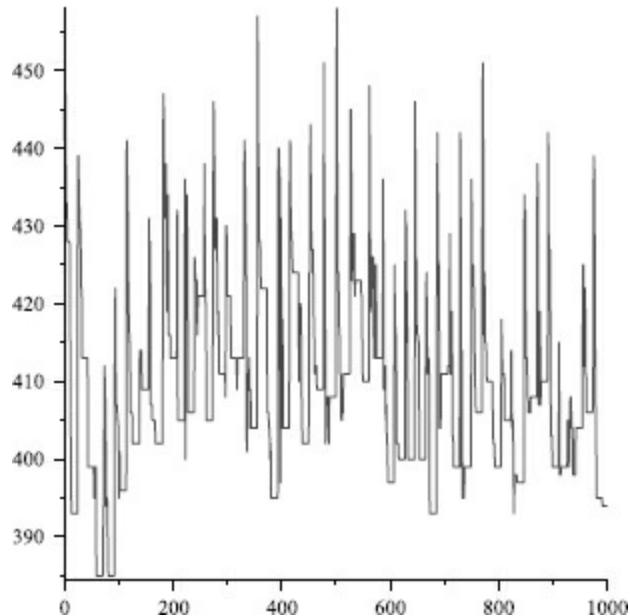
Other possible crossover methods for TSPs will be explored in the exercises. Once a new generation of n solutions has been developed, we need to determine possible mutations to some solutions. Mutations are rare alterations of a solution, meant to add (or preserve) an element of diversity into the population. Typically, the probability a given solution experiences mutation is very small, and the change is only in a few (typically one) elements. Typical mutations are the flipping of a binary variable from 0 to 1 (or 1 to 0) or the swapping of two elements in a permutation.

To generate a genetic algorithm, there are a few parameters that are supplied by the user: n , the size of the population, solution encoding and fitness function, probability distribution associated with crossover point, probability of mutation, selection criteria for the mating population, and the number of generations to produce. An outline of a genetic algorithm is given in Algorithm 15.4.

Algorithm 15.4 Genetic Algorithm: Outline

Randomly choose initial population P_0 of n solutions.
 Let \mathbf{x}_{best} be best solution in P_0 and v_{best} its objective value.
for $k = 1, \dots, \text{MaxIterations}$ **do**
 $M_k \leftarrow \text{GetMatingPopulation}(P_{k-1})$
 $\widehat{P}_k \leftarrow \text{Crossover}(M_k)$
 $P_k \leftarrow \text{Mutate}(\widehat{P}_k)$
 Compare best solution in P_k to \mathbf{x}_{best} and update if necessary.
end for

FIGURE 15.6 Tour distances obtained by genetic algorithm.



■ EXAMPLE 15.7

Let's implement a genetic algorithm to solve our 15-city TSP (15.1). We will use a population of $n = 50$, the crossover operation is the one described earlier in Example 15.6, and we will produce 1000 generations. If we have a probability a given solution is subject to mutation equal to 0.01 and generate the mating population from the top 40% of the parents, we obtain a tour of length 385. The best solutions from each generation are given in Figure 15.6. Note that the best solutions are typically higher than in the other methods because we do not search for local optimal solutions in these cases. If we perform a local search on each generation's best solution, we find a tour of length 374. Doing local search on each generation is reasonable since it allows us to maintain the diversity of the population while still finding good local optima.

Because of its ability to handle sets of solutions, genetic algorithms are among the more popular approaches used to solve difficult discrete optimization problems. For more information on genetic algorithms, the book by D Goldberg [45] and the tutorial by Reeves [74] are excellent sources to refer to.

15.5 GRASP ALGORITHMS

The metaheuristics previously mentioned all began with a feasible solution (or a population of feasible solutions) and iteratively altered it throughout the process. Greedy randomized adaptive search procedures (GRASP) combine a constructive heuristic with local search and were first described by Feo and Resende [31]. The constructive heuristic uses randomization elements to generate an initial solution, the local search approach improves upon this solution, and the entire process is repeated multiple times to search through large portions of the feasible region. Of course, this basic framework provides great flexibility in implementation. A general overview of a GRASP algorithm is given in Algorithm 15.5.

Algorithm 15.5 Greedy Randomized Adaptive Search Procedure (GRASP)

```

 $x_{best} \leftarrow \emptyset$                                 {Initially no best solution}
 $v_{best} \leftarrow -\infty$                             {Value of best solution found}
for  $k = 1, \dots, \text{MaxIterations}$  do
     $x \leftarrow \text{GenerateRandomGreedySolution}()$ 
     $x \leftarrow \text{DoImprovement}(x)$ 
    if  $f(x)$  is better than  $v_{best}$  then
         $x_{best} \leftarrow x$ 
         $v_{best} \leftarrow f(x)$ 
    end if
end for

```

In each iteration, GRASP algorithms first construct an initial solution through a modified greedy approach. In typical greedy algorithms, the next element of a solution is chosen that generates the best choice among all possible selections; for minimization problems, this means the smallest increase, while for maximization this is the greatest increase. Such greedy choices are local by nature, since the current best choice may not be the globally best one, in that it may yield fewer good options later in the algorithm. GRASP algorithms add randomization to this process to generate

different solutions. At each step of the constructive algorithm, a list of possible selection elements is created and one is then selected at random. This *restricted candidate list* (RCL) typically includes only the best candidates for selection; if the RCL contains all possible elements, then we are simply generating a random solution. There are various schemes to generate the RCL, but some of the common ones are as follows:

- Choose the best α elements.
- Choose the top α percent (e.g., top 25%, top 50%, etc.).
- Choose all elements within α percent of the greedy selection.

The parameter α typically does not change throughout the process, but this is no set rule. Once the RCL has been determined, an element from it is randomly selected and added to our current partial solution. The values of the remaining elements, which will be used when the next RCL is constructed, are updated and the process is repeated until a complete solution is obtained. The construction phase is outlined in Algorithm 15.6. Once the constructive phase is complete, we begin the improvement phase. Typically, this involves a basic local search algorithm, but it can include any other method as well, such as simulated annealing or tabu search.

Algorithm 15.6 GRASP GenerateRandomGreedySolution()

```

 $x \leftarrow \emptyset$                                 {Initially empty solution}
 $v_{best} \leftarrow -\infty$                          {Value of best solution found}
while solution  $x$  not complete do
     $RCL \leftarrow \text{GenerateRCL}(\alpha)$ 
     $c \leftarrow \text{RandomSelection}(RCL)$ 
    Add  $c$  to  $x$ 
    UpdateGreedyValues( $x$ )
end while
return  $x$ 

```

■ EXAMPLE 15.8

Consider the following TSP on six cities with distance matrix

$$c = \begin{bmatrix} - & 14 & 23 & 25 & 12 & 15 \\ 14 & - & 12 & 23 & 30 & 13 \\ 23 & 12 & - & 7 & 15 & 28 \\ 25 & 23 & 7 & - & 18 & 15 \\ 12 & 30 & 15 & 18 & - & 20 \\ 15 & 13 & 28 & 15 & 20 & - \end{bmatrix}.$$

Suppose, we have a GRASP algorithm where the RCL is constructed at each

iteration by selecting the top 25% available cities (thus, we have $\alpha = 0.25$). In one iteration of the `GenerateRandomGreedySolution()` procedure, we first randomly select city 5 as our initial city. The closest city to 5 is city 1 with distance 12, and the farthest city from 5 is city 2 with distance 30; hence, we construct our RCL from those cities whose distances are between 12 and $12 + 0.25(30 - 12) = 16.5$, which gives an RCL of {1, 3}. Suppose, we randomly select city 3 from this RCL. Now consider the distances from city 3 to the unvisited cities. The minimum distance is 7 (city 4) and the maximum distance is 28 (city 6), so the RCL consists of those unvisited cities whose distance is between 7 and $7 + 0.25(28 - 7) = 12.25$, which gives {2, 4}. Suppose, we randomly select city 2. In this iteration, the RCL consists of those cities whose distance is between 13 and $13 + 0.25(23 - 13) = 15.5$, which gives {1, 6}. Randomly selecting city 6, our next RCL is {1, 4}, since both cities are the only unvisited cities and each has distance 15 from city 6. If we randomly select city 4, our last city to visit is city 1, and we have generated the tour

$$5 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 4 \rightarrow 1 \rightarrow 5,$$

whose distance is 92. We now run our local search method. If we consider swapping cities 2 and 4 on our tour, we find the better solution

$$5 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 2 \rightarrow 1 \rightarrow 5,$$

which has distance 76 and is, in fact, the optimal solution.

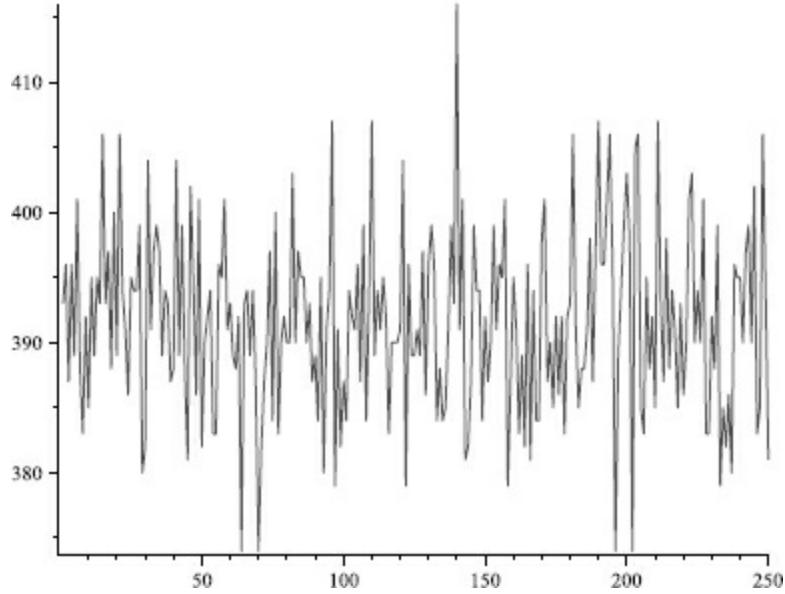
GRASP methods generally perform very well. They often generate optimal or near-optimal solutions to difficult optimization problems, and there have been many applications of GRASP in the literature (see Festa and Resende [36] for an annotated bibliography). A large part of its appeal is its ease of implementation. GRASP algorithms are very easy to construct since there are very few parameters to maintain. Furthermore, they lend themselves naturally to parallel implementations, since the only common component to each iteration is the best solution found. Finally, they can easily be combined with other metaheuristics such as tabu search. This enables us to search various regions of the feasible region both through the randomization component of the constructive phase and through the improvement phase.

■ EXAMPLE 15.9

Let's consider again the 15-city TSP given by the distance matrix (15.1). If we construct a GRASP algorithm where in each iteration the RCL consists of those unvisited cities whose distance from the current city is between the minimum distance and $\min + 0.25(\max - \min)$ and run it for 250 iterations, we find a tour of length 374. The distances of each obtained tour is given in [Figure 15.7](#). Note that most of the obtained tours have distances between 380 and 410, indicating that there are many local optimal solutions with these values, and few solutions whose distance is outside this range.

There are some drawbacks to using GRASP methods. First, if we use the “basic” approach, we always stop once a local optimal solution is found, so diversification comes only through the constructive phase. A large number of iterations are required, therefore, to adequately explore vast amounts of the feasible region. Second, unlike tabu search and genetic algorithms, the basic GRASP does not use any search memory; each iteration is independent of the others. One way that memory can be incorporated into GRASP is to alter the values of α throughout the process. One approach, called *Reactive GRASP*, assumes that at each iteration, the value of α is randomly chosen from a set $A = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$ of possible values. Initially, each α_k has probability $p_k = \frac{1}{m}$ of being selected. Suppose that v_{best} is the current best solution found and A_k is the average value of the solutions generated when $\alpha = \alpha_k$ is used. If we have a maximization problem, let $q_k = \frac{A_k}{v_{\text{best}}}$ and redefine our probabilities p_k by

FIGURE 15.7 Tour distances obtained by GRASP method.



$$p_k = \frac{q_k}{\sum_{j=1}^m q_j}.$$

Thus, those values of α_k that generally produce better solutions will have a higher probability of being used in subsequent iterations. This approach was first suggested by Prais and Ribeiro [72] when studying a problem in TDMA traffic assignment.

Because of its ease of implementation and its effectiveness as either a stand-alone metaheuristic or as part of a hybrid with other metaheuristics, there is an extensive number of papers describing GRASP and its applications. Those interested in learning more about GRASP and its applications should explore the works of Festa and Resende –34, 35, 36—and Resende and Ribeiro [76] for recent advances in the approach.

Summary

In this chapter, we introduced some common metaheuristic techniques that are extensively used to solve difficult real-world problems. They each have their advantages and disadvantages, and so it is important that we as problem solvers have these tools available to us when attempting to find quick solutions to a problem.

EXERCISES

15.1 Run a simulated annealing algorithm on the following TSP instance:

$$\mathbf{c} = \begin{bmatrix} - & 13 & 8 & 15 & 7 & 16 & 19 & 21 \\ 13 & - & 5 & 7 & 14 & 22 & 11 & 14 \\ 8 & 5 & - & 15 & 17 & 9 & 13 & 12 \\ 15 & 7 & 15 & - & 8 & 7 & 9 & 10 \\ 7 & 14 & 17 & 8 & - & 12 & 18 & 11 \\ 16 & 22 & 9 & 7 & 12 & - & 8 & 14 \\ 19 & 11 & 13 & 9 & 18 & 8 & - & 15 \\ 21 & 14 & 12 & 10 & 11 & 14 & 15 & - \end{bmatrix}. \quad (15.2)$$

Indicate your appropriate choice of T_0 and use $\alpha = 0.9$.

15.2 Develop a simulated annealing algorithm for the single machine weighted completion time problem first given in Exercise 5.15. Illustrate your approach by doing five iterations with initial temperature $T_0 = 2500$ and $\alpha = 0.8$, using the data from Exercise 5.15.

15.3 Develop a simulated annealing algorithm for the single machine tardiness problem first given in Exercise 5.16. Illustrate your approach by doing five iterations with initial temperature $T_0 = 75$ and $\alpha = 0.8$, using the data from Exercise 5.16.

15.4 Solve the TSP (15.2) using tabu search. Use the same tabu list and aspiration criteria used in Example 15.3. Illustrate the method by doing five iterations.

15.5 Develop a tabu search algorithm for the single machine weighted completion time problem first given in Exercise 5.15. What is your tabu list and aspiration criteria? Illustrate your approach by doing five iterations, using the data from Exercise 5.15.

15.6 Develop a tabu search algorithm for the single machine tardiness problem first given in Exercise 5.16. What is your tabu list and aspiration criteria? Illustrate your approach by doing five iterations, using the data from Exercise 5.16.

15.7 Design a tabu search algorithm for the set covering problem. Illustrate your approach using the instance given in Exercise 5.5.

15.8 Design a tabu search algorithm for the knapsack problem. Illustrate your approach using the problem

$$\max \quad 18x_1 + 15x_2 + 35x_3 + 25x_4 + 19x_5 + 10x_6 + 40x_7 + 30x_8$$

s.t.

$$6x_1 + 4x_2 + 7x_3 + 4x_4 + 5x_5 + 3x_6 + 10x_7 + 8x_8 \leq 30$$

$$x_i \in \{0, 1\} \quad i \in \{1, 2, \dots, 8\}.$$

15.9 For each of the following pairs of parents in an eight-city TSP, perform the crossover technique given in Example 15.6 to generate their offspring, using a breakpoint location $c = 4$.

(a) $P_1 = (1, 3, 4, 6, 2, 8, 5, 7)$ and $P_2 = (4, 2, 6, 5, 8, 7, 1, 3)$.

(b) $P_1 = (3, 1, 5, 4, 7, 6, 8, 2)$ and $P_2 = (8, 1, 3, 2, 5, 6, 4, 7)$.

15.10 A different crossover approach for TSP uses two crossover points $c_1 < c_2$. The section after c_1 and up to and including c_2 gives an interchange mapping. For example, if we had the parents

- (a) $P_1 = (1, 3, 4, 6, 2, 8, 5, 7)$ and $P_2 = (4, 2, 6, 5, 8, 7, 1, 3)$.
- (b) $P_1 = (3, 1, 5, 4, 7, 6, 8, 2)$ and $P_2 = (8, 1, 3, 2, 5, 6, 4, 7)$.

and let $c_1 = 2, c_2 = 4$, we get the mapping $6 \leftrightarrow 2, 5 \leftrightarrow 1$, which means that in each parent we interchange 6 for 2, 2 for 6, 5 for 1, and 1 for 5, leaving the

remaining elements the same. This generates the offspring

$$O_1 = (4, 3, 2, 1, 5, 6)$$

$$O_2 = (3, 1, 6, 5, 2, 4).$$

Illustrate this crossover approach on the following examples, using the breakpoint locations $c_1 = 3, c_2 = 6$.

$$P_1 = (4, 3, 6, 5, 1, 2)$$

$$P_2 = (3, 5, 2, 1, 6, 4).$$

15.11 Another crossover approach for TSP uses two crossover points $c_1 < c_2$. The section after c_1 and up to and including c_2 for one parent replaces that segment for the other. However, in order to avoid repeated values in the sequence, we first remove the values from the “receiving” parent, shift these “holes” to their proper location, and replace the holes with new

locations. For example, if we had the parents

$$P_1 = (4, 3, 6, 5, 1, 2)$$

$$P_2 = (3, 5, 2, 1, 6, 4)$$

and let $c_1 = 2, c_2 = 4$, the segment from P_1 is $(6, 5)$ and from P_2 it is $(2, 1)$. In Parent 1, we remove elements 1 and 2 and move these holes to positions 3 and 4, generating $(4, 3, H, H, 6, 5)$. Replacing these holes with $(2, 1)$ gives the offspring

$$O_1 = (4, 3, 2, 1, 6, 5).$$

Similar calculations generate the second offspring $O_2 = (3, 2, 6, 5, 1, 4)$. Illustrate this crossover approach in the following examples, using the breakpoint locations $c_1 = 3, c_2 = 6$:

$$P_1 = (4, 3, 6, 5, 1, 2)$$

$$P_2 = (3, 5, 2, 1, 6, 4).$$

This generates the offspring

$$O_1 = (4, 3, 2, 1, 5, 6)$$

$$O_2 = (3, 1, 6, 5, 2, 4).$$

Illustrate this crossover approach in the following examples, using the breakpoint locations $c_1 = 3, c_2 = 6$:

$$P_1 = (4, 3, 6, 5, 1, 2)$$

$$P_2 = (3, 5, 2, 1, 6, 4).$$

15.12 Solve the TSP ([15.2](#)) using GRASP. Be sure to indicate how the RCL is constructed and how a local search is preformed. Illustrate the method by doing five iterations.

15.13 Design a GRASP algorithm for the set covering problem. Be sure to indicate how the RCL is constructed and how a local search is preformed. Illustrate your approach using the instance given in Exercise 5.5.

15.14 Design a GRASP algorithm for the knapsack problem. Be sure to indicate how the RCL is constructed and how a local search is preformed. Illustrate your approach using the problem

$$\begin{aligned}
\max \quad & 18x_1 + 15x_2 + 35x_3 + 25x_4 + 19x_5 + 10x_6 + 40x_7 + 30x_8 \\
\text{s.t.} \quad & 6x_1 + 4x_2 + 7x_3 + 4x_4 + 5x_5 + 3x_6 + 10x_7 + 8x_8 \leq 30 \\
& x_i \in \{0, 1\} \quad i \in \{1, 2, \dots, 8\}.
\end{aligned}$$

15.15 Design a GRASP algorithm for the bin packing problem first given in Exercise 5.17. Be sure to indicate how the RCL is constructed and how a local search is preformed. Illustrate your approach using the instance given in Exercise 5.17.

15.16 The *Overlapping Covers Problem* tries to identify two feasible solutions \mathbf{x}, \mathbf{y} to set covering constraints that have a minimum number of selected variables in common. This nonlinear problem can be stated as

$$\begin{aligned}
\min \quad & \sum_{k=1}^n x_k y_k \\
\text{s.t.} \quad & \sum_{k \in S_j} x_k \geq 1, \quad j \in \{1, \dots, m\} \\
& \sum_{k \in S_j} y_k \geq 1, \quad j \in \{1, \dots, m\} \\
& x_k, y_k \in \{0, 1\}.
\end{aligned}$$

Develop a GRASP algorithm for this problem. Be sure to indicate how the RCL is constructed and how a local search is preformed. Illustrate your method on the constraints from Exercise 5.4.

APPENDIX A

BACKGROUND REVIEW

A.1 BASIC NOTATION

In this section, we review some basic notations used throughout the book.

A **set** is a collection of elements. It can contain a finite number of elements, and infinite number, or no elements at all; in this last case, it is called a **null** or empty set and is denoted by \emptyset . If x is an element of a set S , we denote this by $x \in S$; if x is not in S , we denote this by $x \notin S$. If a set S contains a finite number of elements, then $|S|$ gives the number of elements in S and is called the size of S .

A set T is a **subset** of S if every element x in T is also an element of S and is denoted by $T \subseteq S$. Two sets S and T are equal if $S \subseteq T$ and $T \subseteq S$. If T is a subset of S and there exists an element x that is in S but not in T , then T is a strict subset of S and we denote this by $T \subset S$. Note that, by definition, we have that $\emptyset \subset S$ for every set S .

There are some specific sets that we often use.

R We denote the set of real numbers by \mathbf{R} .

Z The set of integers is denoted by \mathbf{Z} .

S^n For any set of values S , we let S^n denote the set of n -dimensional solutions \mathbf{s} where each $s_k \in S$, $k \in \{1, \dots, n\}$. In particular, we let \mathbf{R}^n denote the vectors of size n where each component is a real number, \mathbf{Z}^n be the set of vectors whose components are all integers, and $\{0, 1\}^n$ denote those n -dimensional vectors whose components are all either 0 or 1.

Set Operations

There are some basic set operations that we now introduce.

Given two sets S and T , the **union** $S \cup T$ of S and T is the set of all elements that belong to either S or T or both. The **intersection** $S \cap T$ of S and T is the set of all elements that belong to both S and T . The **difference** $S - T$ is the set of all elements in S but not in T . The complement \bar{S} of a set S denotes all elements that are not in S ; note that this requires the knowledge of a **universal set** U giving all possible elements.

■ EXAMPLE A.1

Suppose we let

$$S = \{1, 3, 5, 6, 8\}$$

$$T = \{2, 3, 4, 7, 8\}.$$

If we also define the universal set $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$, then we have the following sets:

$$S \cup T = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$S \cap T = \{3, 8\}$$

$$S - T = \{1, 5, 6\}$$

$$T - S = \{2, 4, 7\}$$

$$\bar{S} = \{2, 4, 7\}$$

$$\bar{T} = \{1, 5, 6\}.$$

Minimums and Argmins

Often, we will assign function values $f(x)$ to each element x of a set S and try to identify, when it exists, the minimum and/or maximum value. Other times, we are interested in which element generated these minimum and maximum values.

Given a finite set S and a function f defined for all elements of S , the **argmin** of f over S is the element $x \in S$ that provides the minimum value of $f(x)$ over $x \in S$; this is denoted by

$$\arg \min_{x \in S} f(x);$$

similarly, the **argmax** of f over S is the element $x \in S$ that provides the maximum value of $f(x)$ over $x \in S$ and is denoted by

$$\arg \max_{x \in S} f(x).$$

■ EXAMPLE A.2

Suppose we let $S = \{1, 2, 6, 7\}$ and $f(x) = (x - 3)^2$. Thus, we have the potential function values

$$\{f(1), f(2), f(4), f(7)\} = \{4, 1, 9, 16\}.$$

We then have

$$\arg \min_{x \in S} f(x) = 2,$$

$$\arg \max_{x \in S} f(x) = 7,$$

since these values generate the minimum and maximum values, respectively.

A.2 GRAPH THEORY

A graph G consists of a set V of *vertices* or *nodes* that are linked pairwise. If the links (i, j) are not ordered pairs, then $(i, j) = (j, i)$ is called an *edge* of G and the set of edges is E ; in this case, our graph G is called an *undirected graph* and is denoted by $G = (V, E)$. If the links are ordered pairs, then (i, j) is an *arc* of the graph and the set of arcs of G is denoted by A . G is then a *directed graph* and is denoted by $G = (V, A)$. Each arc $(i, j) \in A$ is said to be *oriented* from i to j , or simply have an *orientation*, where node i is the *tail* and node j is the *head* of the arc (i, j) . We typically denote the number of nodes/vertices of G as $|V| = n$ and the number of edges/arcs of G to be m .

Undirected Graphs

In this section, we define a variety of terms associated with undirected graphs $G = (V, E)$. The next section provides similar terms for directed graphs.

An edge $(i, j) \in E$ is said to be *incident* or *adjacent* to both nodes i and j . The degree of a node i , denoted by $d(i)$, is equal to the number of edges incident to i in G . We often use the notation

$$A(i) = \{j \in V : (i, j) \in E\}$$

to denote those vertices incident to the same edge as i , although sometimes, depending on the context, we will use

$$A(i) = \{(i, j) \in E : j \in V\}.$$

In either case, we have $d(i) = |A(i)|$.

A graph $G' = (V', E')$ is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$. The subgraph $G' = (V', E')$ is *induced* by V' if E' contains exactly those edges $(i, j) \in E$ where both $i, j \in V'$.

A **walk** in G is a subgraph of G consisting of a sequence of nodes $i_1 - i_2 - \dots - i_{k-1} - i_k$ where each $(i_j, i_{j+1}) \in E$; a node may be repeated, but not an edge. A **path** is a walk where the nodes are not repeated. A **cycle** is a path such that $i_1 = i_k$. An **acyclic graph** is a graph that contains no cycle.

Two nodes i, j are **connected** if the graph contains a path from i to j . A graph is **connected** if every pair of nodes is connected. A **component** of a graph is a maximally connected subgraph.

Directed Graphs

Many of the above terms that were given for undirected graphs can be extended to directed graphs, with appropriate modifications.

A directed arc $(i, j) \in A$ is said to be **incident** to both i and j . j is **adjacent** to i . Arc (i, j) is an **outgoing arc** from i and an **incoming arc** to j .

The **indegree** of a node is the number of its incoming arcs. The **outdegree** is the number of its outgoing arcs. The **degree** of a node is its indegree plus its outdegree. The **arc adjacency list** $A(i) = \{(i, j) \in A : j \in V\}$, or the set of its outgoing arcs. Sometimes, we also use $A(i) = \{j : (i, j) \in A\}$, the **node adjacency list**. The use of $A(i)$ should be clear from its context.

A directed graph $G' = (V', A')$ is a **subgraph** of G if $V' \subseteq V$ and $A' \subseteq A$.

A **walk** in G is a subgraph of G consisting of a sequence of nodes and arcs $i_1 - a_1 - i_2 - a_2 - \dots - i_{k-1} - a_{k-1} - i_k$ where each $a_j = (i_j, i_{j+1})$ or $a_j = (i_{j+1}, i_j)$ (i.e., orientation of arc is ignored) and a node may be repeated, but not an arc. A **path** is a walk where the nodes are not repeated. A **cycle** is a path such that $i_1 = i_k$. In each case, the arcs are typically left off the list. **Directed walks**, **directed paths**, and **directed cycles** follow all orientations of the arcs. An **acyclic graph** is a graph that contains no directed cycle.

Two nodes i, j are **connected** if the graph contains a path from i to j . A graph is **connected** if every pair on nodes is connected. A **component** of a graph is a maximally connected subgraph. A connected graph is **strongly connected** if between every two nodes i, j there exists a directed path from one to the other.

Trees and Forests

For both directed and undirected graphs, the notion of a tree is defined. These special subgraphs appear often when discussing optimization problems on graphs. The definitions given here apply to both directed and undirected graphs.

A **forest** is a graph that contains no cycle. If the graph is connected and contains no cycles, it is a **tree**. A **spanning tree** of a graph G is a subgraph of G such that (a) it is a tree and (b) every node in G is contained in the subgraph.

Trees and forests have special properties that we can exploit.

1. The number of arcs/edges in a forest equals $n - (\# \text{ components})$. Since a tree has one component, the number of arcs/edges in a tree is $n - 1$.
2. There exist at least two nodes in a tree whose degree is equal to 1. Such nodes are called **leaf nodes**.
3. In a tree, there exists a unique path between every pair of nodes i, j .
4. Removing an arc/edge from a tree separates the graph into a forest with two components.

A.3 LINEAR ALGEBRA

This appendix provides a review of basic matrix and linear algebra, especially topics useful in the study of linear optimization models.

Matrices and Vectors

Let us first give a formal definition of a matrix and a vector.

Matrix A *matrix* is any rectangular array of numbers. The number in the

*i*th row and *j*th column of a matrix A is called the (i, j) th element of A and is denoted by a_{ij} . In this book, matrices will always be denoted by capital letters.

Order of Matrix We say that a matrix A is an $m \times n$ matrix if it has m rows and n columns. We refer to $m \times n$ as the *order* of the matrix. Such a matrix is typically written as

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Vector A *vector* is a matrix of order $m \times 1$. It is sometimes referred to as a column vector. A matrix of order $1 \times m$ is sometimes referred to as a row vector. In this book, all vectors are column vectors and will be denoted by **bold lowercase letters**. The dimension of a vector is the number of rows.

We will also note that a matrix $A = [a_{ij}]$ is one where the (i, j) th entry of A is a_{ij} . If a matrix has an equal number of rows and columns ($m = n$), it is referred to as a **square matrix**. A 1×1 matrix is a number or *scalar*.

■ EXAMPLE A.3

The following are examples of matrices:

$$\begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 6 & 7 \\ 1 & -8 \\ 4 & 3 \end{bmatrix},$$

while a (column) vector and a row vector could look like

$$\begin{bmatrix} 1 \\ 0 \\ 8 \end{bmatrix}, \quad [-1 \ 4].$$

For simplicity, we will often write a vector using parenthesis instead of brackets. When we do, **it is still a column vector**. Hence, the vector $\mathbf{v} =$

$(1, 0, 8)$ is the same as

$$\mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ 8 \end{bmatrix}$$

but not the same as

$$\begin{bmatrix} 1 & 0 & 8 \end{bmatrix} = \mathbf{v}^T.$$

Two matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ are **equal** if and only if A and B are of the same order and $a_{ij} = b_{ij}$ for all i and j .

Note that a matrix A can (and often is) viewed as a collection of n (column) vectors. In fact, one will often write

$$A = [a_{ij}] = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \dots \quad \mathbf{a}_n].$$

We can also view a matrix A as a collection of m row vectors.

Special Matrices and Vectors

Some vectors and matrices appear so often that they are specially designated. For each of these matrices, the order is assumed to be inferred by its context.

The **null vector** or **zero vector** $\mathbf{0}$ designates a vector containing all 0's, namely,

$$\mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Similarly, we define the **zero matrix** 0 as the matrix of all 0's.

The vector \mathbf{e} is a vector where every element is a 1,

$$\mathbf{e} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

and the vector \mathbf{e}_k is a vector where the k th element is 1 and the rest are 0; for

example,

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbf{e}_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

The square matrix I is one where $a_{ii} = 1$ for all i and $a_{ij} = 0$ if $i \neq j$. Such a matrix is referred to as the **identity matrix**. Note that each column of I is a vector \mathbf{e}_k .

Matrix and Vector Operations

In this section, we describe the various arithmetic operations on matrices and vectors one typically performs.

Scalar Multiplication Given any matrix A and any scalar k , the matrix $C = kA$ is obtained by multiplying each element a_{ij} of A by k ; that is, $c_{ij} = ka_{ij}$. For example, given the matrix

$$A = \begin{bmatrix} 6 & 7 \\ 1 & -8 \\ 4 & 3 \end{bmatrix}$$

and the value $k = 2$, we get

$$kA = 2A = \begin{bmatrix} 12 & 14 \\ 2 & -16 \\ 8 & 6 \end{bmatrix}.$$

Scalar multiplication also works for vectors; if \mathbf{v} is a n -dimensional vector and k is any scalar, then

$$k\mathbf{v} = \begin{bmatrix} kv_1 \\ kv_2 \\ \vdots \\ kv_n \end{bmatrix}.$$

Addition Let A and B be two matrices of the same order ($m \times n$). The matrix $C = A + B$ is one where $c_{ij} = a_{ij} + b_{ij}$. If A and B are not of the same order,

they cannot be added together. For example, if

$$A = \begin{bmatrix} 1 & -2 \\ 3 & 0 \\ -1 & 3 \end{bmatrix} \text{ and } B = \begin{bmatrix} 6 & 7 \\ 1 & -8 \\ 4 & 3 \end{bmatrix},$$

then

$$C = A + B = \begin{bmatrix} 1 & -2 \\ 3 & 0 \\ -1 & 3 \end{bmatrix} + \begin{bmatrix} 6 & 7 \\ 1 & -8 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 7 & 5 \\ 4 & -8 \\ 3 & 6 \end{bmatrix}.$$

Please note that $A + B = B + A$.

Multiplication In order to multiply two matrices A and B , we need to be more careful. First, we can only form the product $C = AB$ if A is an $m \times r$ matrix and B is an $r \times n$ matrix; if this is not the case, the product is undefined. The order of the resulting matrix C is $m \times n$. To form the (i, j) th entry of C , we let

$$c_{ij} = \sum_{k=1}^r a_{ik} b_{kj}.$$

For example, given matrices

$$A = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & 1 \\ 0 & 1 \\ 1 & 2 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}, \quad D = [4 \ -2 \ 1],$$

we have

$$\begin{aligned}
AB &= \begin{bmatrix} 1 & 0 & -1 \\ 2 & 1 & 2 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & 1 \\ 1 & 2 \end{bmatrix} \\
&= \begin{bmatrix} 1(3) + 0(0) + (-1)(1) & 1(1) + (0)(1) + (-1)(2) \\ 2(3) + 1(0) + (2)(1) & 2(1) + 1(1) + (2)(2) \end{bmatrix} \\
&= \begin{bmatrix} 2 & -1 \\ 8 & 7 \end{bmatrix},
\end{aligned}$$

$$\begin{aligned}
BA &= \begin{bmatrix} 3 & 1 \\ 0 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 1 & 2 \end{bmatrix} \\
&= \begin{bmatrix} 3(1) + (1)(2) & 3(0) + (1)(1) & 3(-1) + (1)(2) \\ 0(1) + (1)(2) & 0(0) + (1)(1) & 0(-1) + (1)(2) \\ 1(1) + (2)(2) & 1(0) + (2)(1) & 1(-1) + (2)(2) \end{bmatrix}
\end{aligned}$$

$$= \begin{bmatrix} 5 & 1 & -1 \\ 2 & 1 & 3 \\ 5 & 2 & 3 \end{bmatrix},$$

$$\begin{aligned}
DC &= [4 \quad -2 \quad 1] \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} = 4(1) + (-2)(0) + (1)(3) \\
&= 8,
\end{aligned}$$

$$\begin{aligned}
CD &= \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} [4 \quad -2 \quad 1] \\
&= \begin{bmatrix} 1(4) & 1(-2) & (1)(1) \\ (0)(4) & (0)(-2) & (0)(1) \\ (3)(4) & (3)(-2) & (3)(1) \end{bmatrix} \\
&= \begin{bmatrix} 4 & -2 & 1 \\ 0 & 0 & 0 \\ 12 & -6 & 3 \end{bmatrix}.
\end{aligned}$$

As noted above, the order of the multiplication is important because

generally $AB = BA$, even when both operations are defined.

It is important to note that, no matter the matrix A , we have

$$AI = IA = A,$$

where the identity matrices may be of different orders. In addition, we have

$$A0 = 0A = 0.$$

Transpose Given an $m \times n$ matrix $A = [a_{ij}]$, the transpose of A , denoted by A^T , is the $n \times m$ matrix where $a^T_{ij} = a_{ji}$. For example,

$$A = \begin{bmatrix} 1 & -2 \\ 3 & 0 \\ -1 & 3 \end{bmatrix} \implies A^T = \begin{bmatrix} 1 & 3 & -1 \\ -2 & 0 & 3 \end{bmatrix}.$$

Vector Dot Product Given two n -dimensional vectors \mathbf{u} and \mathbf{v} , we can define the **dot product** $\mathbf{u} \cdot \mathbf{v}$ as

$$\mathbf{u} \cdot \mathbf{v} = \sum_{j=1}^n u_j v_j;$$

this definition is also equivalent to

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \mathbf{v}^T \mathbf{u},$$

which is written in terms of matrix multiplication. Note that the result of this operation is a scalar and not a vector. We also use this calculation to denote the norm $\|\mathbf{x}\|$ of a vector as

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}} = \sqrt{\sum_{j=1}^n x_j^2}.$$

Determinant If A is a square $n \times n$ matrix, its **determinant** is a scalar associated with A . If A is a 2×2 matrix, then

$$\det(A) = \det \left(\begin{bmatrix} a & b \\ c & d \end{bmatrix} \right) = ad - bc.$$

To calculate the determinant of larger matrices, we need the notion of a minor.

Determinant If A is an $n \times n$ matrix, then for any i, j the ij th **minor** A_{ij}

of A is the $(n - 1) \times (n - 1)$ submatrix of A obtained by deleting the i th row and j th column of A .

■ EXAMPLE A.4

If

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

Then

$$A_{13} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}.$$

To calculate the determinant of A , we first choose an index $1 \leq i \leq n$ and then use

$$\det(A) = \sum_{j=1}^m (-1)^{i+1} a_{ij} \det(A_{ij}).$$

■ EXAMPLE A.5

If

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

then, if we choose $i = 1$,

$$\begin{aligned} \det(A) &= (-1)^{1+1}(1)\det\left(\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}\right) + (-1)^{2+1}(2)\det\left(\begin{bmatrix} 4 & 6 \\ 7 & 9 \end{bmatrix}\right) \\ &\quad + (-1)^{3+1}(3)\det\left(\begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix}\right) \\ &= 1(45 - 48) - (2)(36 - 42) + (3)(32 - 35) \\ &= -3 - (2)(-6) + (3)(-3) \\ &= 0. \end{aligned}$$

For a fixed vector \mathbf{a} and scalar b , we denote

$$S = \{\mathbf{x} : \mathbf{a}^T \mathbf{x} = b\}$$

to be a **hyperplane** in \mathbb{R}^n . If $n = 2$, this is simply a line. The vector \mathbf{a} is called the **normal vector** of the hyperplane and lies perpendicular to it.

Systems of Equations and Elementary Row Operations

One of the main uses of matrices arises when trying to solve a system of equations. Consider the system of equations

$$(A.1) \quad x + y + z = 4$$

$$(A.2) \quad 2x + y + z = 5$$

$$(A.3) \quad x + 2y + 3z = 8.$$

This system can be written in matrix–vector form as

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix}.$$

We can mimic the operations used to solve such a system through **elementary row operations** on a matrix.

Given a matrix A , the following operations are known as **elementary row operations** on A :

1. Multiply any single row of A by a constant k .
2. Interchange any two rows of A .
3. Add a constant multiplier of one row of A to another row.

Two matrices A and B are **row equivalent** if one can be obtained from the other using a finite sequence of elementary row operations.

■ EXAMPLE A.6

Given the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix},$$

suppose we perform the elementary row operation # 3 on row 2 by taking $(\text{row 2}) \leftarrow (\text{row 2}) - 2(\text{row 1})$, which generates the row equivalent matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 1 & 2 & 3 \end{bmatrix}.$$

If we replace row 3 by $(\text{row 3}) - (\text{row 1})$, we get the row equivalent matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 1 & 2 \end{bmatrix}.$$

If we now replace row 3 by $(\text{row 3}) + (\text{row 2})$ and then row 2 by $-(\text{row 2})$, we get the row equivalent matrix

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Notice the form of matrix B in the above example. In each row, the first nonzero element occurs “to the right” of the first nonzero element of the previous row. Such matrices are known as *row-echelon matrices*.

Matrix A is *row-echelon* if it satisfies the following properties:

1. Every row of A consisting of all zero elements is below all rows with at least one nonzero element.
2. Each row of A contains a nonzero element, the first nonzero element occurs in column “to the right” of the first nonzero element in the previous row. This first nonzero element in a row is known as the **leading entry** of the row.

Given a system of equations with n constraints and n variables written as $Ax = \mathbf{b}$, we can solve this system by considering the *augmented matrix* $[A|\mathbf{b}]$ and then converting this to a row equivalent row-echelon matrix.

■ EXAMPLE A.7

Let's solve the system of equations

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix}.$$

We first augmented the matrix

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 1 & 1 & 5 \\ 1 & 2 & 3 & 8 \end{array} \right],$$

where the last column is the right-hand-side vector \mathbf{b} . If we convert this matrix to a row equivalent row-echelon matrix, we get

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 1 & 1 & 3 \\ 0 & 0 & 1 & 1 \end{array} \right],$$

which corresponds to the system of equations

$$(A.4) \quad x + y + z = 4$$

$$(A.5) \quad y + z = 3$$

$$(A.6) \quad z = 1,$$

whose solution is $x = 1$, $y = 2$, $z = 1$. Notice that if we have done the row operations (row 1) \leftarrow (row 1) - (row 2) and then row 2 \leftarrow (row 2) - (row 3), we end up with the row-equivalent matrix

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{array} \right]$$

corresponding to the system

$$(A.7) \quad x = 1$$

$$(A.8) \quad y = 2$$

$$(A.9) \quad z = 1.$$

Notice that, in the above example, we generated a row-echelon matrix where

each row's leading entry had value 1 and was also the only nonzero element of that column. Such a matrix is a **reduced echelon matrix**.

Inverse Matrix

We know that, for any real number $k \neq 0$, there exists a real number k^{-1} such that $kk^{-1} = k^{-1}k = 1$. Is there a similar property for matrices?

Matrix Inverse Given a square $n \times n$ matrix A , the matrix A^{-1} is the matrix satisfying

$$AA^{-1} = A^{-1}A = I.$$

A^{-1} is called the *inverse matrix* of A .

■ EXAMPLE A.8

The inverse of the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

is

$$A^{-1} = \begin{bmatrix} -1 & 1 & 0 \\ 5 & -2 & -1 \\ -3 & 1 & 1 \end{bmatrix}$$

Since $AA^{-1} = A^{-1}A = I$. However, the matrix

$$B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

has no inverse. To see this, suppose B^{-1} exists and is

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Then, $BB^{-1} = I$ generates the following system of equations, one for each element of the resulting matrices:

$$(A.10) \quad a + b = 1$$

$$(A.11) \quad c + d = 0$$

$$(A.12) \quad a + b = 0$$

$$(A.13) \quad c + d = 1,$$

which are clearly inconsistent.

The preceding example shows that not every square matrix A has an inverse. If a matrix has an inverse, it is called **invertible**.

Some important properties of invertible matrices are as follows:

1. Given an invertible matrix A , it has a unique inverse matrix A^{-1} .
2. Given an invertible matrix A , then A^{-1} is invertible and

$$(A^{-1})^{-1} = A.$$

3. If A and B are both invertible $n \times n$ matrices, then AB is invertible and

$$(AB)^{-1} = B^{-1}A^{-1}.$$

4. If A is invertible, then, for any vector $\mathbf{b} \in \mathbb{R}^n$, the system of equations

$$A\mathbf{x} = \mathbf{b}$$

has the unique solution

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

5. A is invertible if and only if it is row equivalent to I . Thus, one way to compute A^{-1} is to begin with the augmented matrix $[A|I]$ and perform elementary row operations until you obtain the form $[I|A^{-1}]$.

6. If A is a 2×2 invertible matrix, then

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

Linear Independence and Rank

Often, we are interested in the case where one vector can be written in terms of others. This is what we are asking when we solve systems of equations, namely, whether the right-hand-side vector \mathbf{b} can be written in terms of the columns of A .

Linear Combination A linear combination of the vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ is any vector of the form

$$\sum_{j=1}^k c_j \mathbf{v}_j = c_1 \mathbf{v}_1 + c_2 \mathbf{v}_2 + \cdots + c_k \mathbf{v}_k,$$

where c_1, \dots, c_k are scalars.

■ EXAMPLE A.9

Given the vectors

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix},$$

if we set $c_1 = 1, c_2 = 2, c_3 = 1$, we get

$$1 \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 8 \end{bmatrix}.$$

Clearly, the zero vector $\mathbf{0}$ is a linear combination of any finite set of vectors since we can set $c_k = 0$ for all k . An interesting question is whether there are other values of c_k that generate $\mathbf{0}$.

Linear Independence A set $V = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ in \mathbb{R}^n are linearly independent if the only linear combination of $\mathbf{v}_1, \dots, \mathbf{v}_k$ that equals $\mathbf{0}$ is $c_1 = \cdots = c_k = 0$.

A set V of vectors that is not linearly independent is **linearly dependent**.

We often view linear independence in terms of a corresponding matrix.

Rank The rank of a matrix A is the number of columns of A that are linearly independent.

It turns out that the rank of A is also equivalent to the number of rows of A that are linearly independent. To determine the rank of matrix A , we employ

elementary row operations to reduce A to a reduced echelon matrix A' ; the rank of A is then the number of leading entries in A' .

■ EXAMPLE A.10

Given the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 4 & 0 \\ 2 & 1 & 1 & 5 & 1 \\ 1 & 2 & 3 & 8 & 7 \end{bmatrix},$$

its reduced echelon form is

$$A' = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 1 & 2 \end{bmatrix}.$$

Since there are three leading entries, the rank of A is 3. This indicates that there is a subset of three linearly independent columns, which can be identified by the columns containing the leading elements and also that all three rows are linearly independent.

It is important to note that **the rank of a matrix is unaffected by permutations of the rows and/or columns.**

■ EXAMPLE A.11

Consider the matrix

$$A = \begin{bmatrix} 1 & 1 & 1 & 4 & 0 \\ 2 & 1 & 1 & 5 & 1 \\ 1 & 2 & 3 & 8 & 7 \end{bmatrix},$$

which has rank of 3. If we interchange rows 1 and 3 and columns 2 and 4, we generate the matrix

$$A_1 = \begin{bmatrix} 1 & 8 & 3 & 2 & 7 \\ 2 & 5 & 1 & 1 & 1 \\ 1 & 4 & 1 & 1 & 0 \end{bmatrix},$$

we still have $\text{rank}(A_1) = 3$.

We can tie many of the above ideas together.

Given an $n \times n$ matrix A , the following statements are equivalent:

1. A^{-1} exists.
 2. $\det(A) \neq 0$.
 3. Rank of A is n .
 4. The system of equations $A\mathbf{x} = \mathbf{b}$ has a unique solution for every vector $\mathbf{b} \in \mathbb{R}^n$.
-

REFERENCES

1. Abrara, J. “Applying integer linear programming to the fleet assignment problem,” *Interfaces*, Vol. 19 no. 4 (1989), pp. 20–28.
2. Ahuja, R.K., Magnanti, T.L., and Orlin, J.B., *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, New Jersey (1993).
3. Applegate, D.L., Bixby, R.E., Chvátal, V., and Cook, W.J., *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, New Jersey (2006).
4. Balakrishnan, A., Magnanti, T.L., and Mirchandani, P., “Designing hierarchical survivable networks,” *Operations Research*, Vol. 46, no. 1 (1998), pp. 116–136.
5. Balas, E., Ceria, S., and Cornuéjols, G. “A lift-and-project cutting plane algorithm for mixed 0–1 programs,” *Mathematical Programming*, Vol. 58 (1993), pp. 295–324.
6. Barnhart, C., Belobaba, P., and Odoni, A.R., “Applications of operations research in the air transport industry,” *Transportation Science*, Vol. 37, no. 4 (2003), pp. 368–391.
7. Barnhart, C. and Cohn, A., “Airline schedule planning: accomplishments and opportunities,” *Manufacturing & Service Operations Management*, Vol. 6, no. 1 (2004), pp. 3–22.
8. Bazargan, M., *Airline Operations and Scheduling*, Ashgate Publishing, Vermont (2004).
9. Bazzara, M.S., Sherali, H.D., and Shetty, C.M., *Nonlinear Programming: Theory and Algorithms*, John Wiley & Sons, Inc., New Jersey (2006).
10. Bean, J.C., Noon, C.E., Ryan, S.M., and Salton, G.J., “Selecting tenants in a shopping mall,” *Interfaces*, Vol. 18, no. 2 (1988), pp. 1–9.
11. Beale, E.M.L., “Cycling in the dual simplex method,” *Naval Research Logistics Quarterly*, Vol. 2, no. 4 (1955), pp. 269–276.
12. Beasley, J.E., Krishnamoorthy, M., Sharaiha, Y.M., and Abramson, D., “Scheduling aircraft landings—the static case,” *Transportation Science*, Vol. 34, no. 2 (2000), pp. 180–197.

13. Bertsekas, D.P., *Network Optimization: Continuous and Discrete Models*, Athena Scientific, Massachusetts (1998).
14. Bertsimas, D.P. and Tsitsiklis, J.N., *Introduction to Linear Optimization*, Dynamic Ideas and Athena Scientific, Massachusetts (1997).
15. Bixby, R.E., “Implementing the simplex method: the initial basis,” *ORSA Journal on Computing*, Vol. 4 no. 3, (1992), pp. 267–284.
16. Bixby, R. E. and Lee, E.K., “Solving a truck dispatching scheduling problem using branch-and-Cut,” *Operations Research*, Vol. 46, no. 3 (1998), pp. 355–367.
17. Bland, R.G., “New finite pivoting rules for the simplex method,” *Mathematics of Operations Research*, Vol. 5 (1977), pp. 103–107.
18. Bloemhof-Ruwaard, J.M., Salomon, M., and Van Wassenhove, L.N., “The capacitated distribution and waste disposal problem,” *European Journal of Operational Research*, Vol. 88 (1996), pp. 490–503.
19. Blum, C. and Roli, A., “Metaheuristics in combinatorial optimization: overview and conceptual comparison,” *ACM Computing Surveys*, Vol. 35, no. 3 (2003), pp. 268–308.
20. Boykin, R.R., “Optimizing chemical production at Monsanto,” *Interfaces*, Vol. 15, no. 1 (1985), pp. 88–95.
21. Bradley, S.P., Hax, A.C., and Magnanti, T.L., *Applied Mathematical Programming*, Addison–Wesley Publishing, Massachusetts (1977).
22. Carino, H.F. and LeNoir, C.H., “Optimizing wood procurement in cabinet manufacturing,” *Interfaces*, Vol. 18, no. 2 (1988), pp. 10–19.
23. Chvátal, V., “Edmonds polytopes and a hierarchy of combinatorial problems,” *Discrete Mathematics*, Vol. 4 (1973), pp. 305–337.
24. Cornuéjols, G., “Revival of the Gomory cuts in the 1990s,” *Annals of Operations Research*, Vol. 149 (2007), pp. 63–66.
25. Current, J., Daskin, M., and Schilling, D., “Discrete network location models,” in *Facility Location: Applications and Theory*, Z. Drezner and H.W. Hamacher (eds), Springer New York (2001).
26. Dantzig, G.B., “Linear programming,” in *History of Mathematical Programming: A Collection of Personal Reminiscences*, J.K. Lenstra, A.H.G. Rinnooy Kan, and A. Schrijver (eds), Elsevier Science Publishers B.V.,

Amsterdam (1991).

27. Dantzig, G.B. and Ramer, J., “The truck dispatching problem,” *Management Science*, Vol. 6 (1959), pp. 80–91.
28. Degraeve, Z. and Schrage, L., “A tire production scheduling system for Bridge-stone/Firestone Off-The-Road,” *Operations Research*, Vol. 45, no. 6 (1997), pp. 789–796.
29. Dowsland, K., “Simulated annealing,” in *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford (1993).
30. Eppen, G.D., Gould, F.J., Schmidt, C.P., Moore, J.H., and Weatherford, L.R., *Introductory Management Science*, Prentice Hall, New Jersey (1998).
31. Feo, T.A. and Resende, M.G.C., “A probabilistic heuristic for a computationally difficult set covering problem,” *Operations Research Letters*, Vol. 9 (1989), pp. 67–71.
32. Ferguson, A.R. and Dantzig, G.B., “The problem of routing aircraft,” *Aeronautical Engineering Review*, Vol. 14 (1956).
33. Ferguson, A.R. and Dantzig, G.B., “The allocation of aircraft to routes—an example of linear programming under uncertain demand,” *Management Science*, Vol. 3 no. 3, (1956), pp. 45–73.
34. Festa, P. and Resende, M.G.C., “GRASP: basic components and enhancements,” in *Global Optimization: Theoretical Foundations and Applications*, A. Abraham, A.-E. Hassanien, and P. Siarry (eds), Springer, 2008.
35. Festa, P. and Resende, M.G.C., “An annotated bibliography of GRASP, Part I: Algorithms,” *International Transactions in Operational Research*, Vol. 16 (2009), pp. 1–24.
36. Festa, P. and Resende, M.G.C., “An annotated bibliography of GRASP, Part II: Applications,” *International Transactions in Operational Research*, Vol. 16 (2009), pp. 131– 172.
37. Fischetti, M., Glover, F., and Lodi, A., “The feasibility pump,” *Mathematical Programming, Series A*, Vol. 104 (2005), pp. 91–104.
38. Forrest, J.J.H. and Goldfarb, D., “Steepest-edge simplex algorithms for linear programming,” *Mathematical Programming*, Vol. 57 (1992), pp. 341–

39. Frank, C.R., "A note on the assortment problem," *Management Science*, Vol. 11 (1965), pp. 724–726.
40. Garey, M. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Co., New York (1979).
41. Gicquel, C., Miégeville, N., Minoux, M., and Dallery, Y., "Optimizing glass coating lines: MIP models and valid inequalities," *European Journal of Operational Research*, Vol. 202 (2010), pp. 747–755.
42. Glover, F., "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, Vol. 13 (1986), pp. 533–549.
43. Glover, F. and Laguna, M., "Tabu search," in *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford (1993).
44. Glover, F. and Laguna, M., *Tabu Search*, Kluwer Academic Publishers, Boston, MA (1997).
45. Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA (1989).
46. Golden, B., Raghavan, S., and Wasil, E., *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer, New York (2008).
47. Goldfarb, D. and Reed, J.K., "A practical steepest-edge simplex algorithm," *Mathematical Programming*, Vol. 12 (1977), pp. 361–371.
48. Gomory, R.E., "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, Vol. 64 (1958), pp. 275–278.
49. Gomory, R.E., "An algorithm for integer solutions to linear programs," in *Recent Advances in Mathematical Programming*, R. Graves and P. Wolfe (eds), McGraw-Hill, New York (1963), pp. 269–302.
50. Gopalakrishnan, B. and Johnson, E.L., "Airline crew scheduling: state-of-the-art," *Annals of Operations Research*, Vol. 140 (2005), pp. 305–337.
51. Gopalan, R. and Talluri, K.T., "The aircraft maintenance routing problem," *Operations Research*, Vol. 46 (1998), no. 2, pp. 260–271.
52. Gu, Z., Nemhauser, G.L., and Savelsbergh, M.W.P., "Lifted cover

inequalities for 0–1 integer programs: computation,” *INFORMS Journal on Computing*, Vol. 10, no. 4 (1998), pp. 427–437.

53. Hane, C.A., Barnhart, C., Johnson, E.L., Marsten, R.E., Nemhauser, G.L., and Sigismondi, G., “The fleet assignment problem: solving a large-scale integer program,” *Mathematical Programming*, Vol. 70, no. 2 (1995), pp. 211–232.

54. Hansen, P., “The steepest ascent mildest descent heuristic for combinatorial programming,” in Proceedings of *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy (1986).

55. Hansen, P. and Wendell, R., “A note on airline commuting,” *Interfaces*, Vol. 11, no. 1 (1982), pp. 85–87.

56. Harris, P.M.J., “Pivot selection methods of the Devex LP code,” *Mathematical Programming*, Vol. 5 (1973), pp. 1–28.

57. Henderson, D., Jacobson, S., and Johnson, A., “The theory and practice of simulated annealing,” in *Handbook of Metaheuristics*, Springer (2003), pp. 287–320.

58. Hillier, M.S. and Bradeau, M.L., “Optimal component assignment and board grouping in printed circuit board manufacturing,” *Operations Research*, Vol. 46, no. 5 (1998), pp. 675–689.

59. Hoffman, A.J. and Kruskal, J.B., “Integral boundary points of convex polyhedra,” in *Linear Inequalities and Related Systems*, H.W. Kuhn and A.W. Tucker (eds), Princeton University Press, Princeton, NJ (1956), pp. 223–246.

60. Holland, J.J., *Adaptation in Natural and Artificial Systems*, the University of Michigan Press, Ann Arbor, MI (1975).

61. Kirkpatrick, S., Gelatt, Jr., C.D., and Vecchi, M.P., “Optimization by simulated annealing”, *Science*, Vol. 220 (1983), pp. 671–680.

62. Katok, E. and Ott, D., “Using mixed-integer programming to reduce label changes in the Coors aluminum can plant,” *Interfaces*, Vol. 30, no. 2 (2000), pp. 1–12.

63. Marchand, H. and Wolsey, L.A., “Aggregation and mixed integer rounding to solve MIPs,” *Operations Research*, Vol. 49 (2001), pp. 363–371.

64. Metropolis, N., Rosenbluth, A.W., Teller, A.H., and Teller, E., “Equation

- of state calculation by fast computing machines,” *Journal of Chemical Physics*, Vol. 21 (1953), pp. 1087– 1091.
65. Miller, C.E., Tucker, A.W., and Zemlin, R.A., “Integer programming formulations and traveling salesman problems,” *Journal of the ACM*, Vol. 7 (1960), pp. 326–329.
66. Nash, S.G. and Sofer, A., *Linear and Nonlinear Programming*, McGraw-Hill, New York (1996).
67. Nemhauser, G.L. and Trick, M.A., “Scheduling a major college basketball conference,” *Operations Research*, Vol. 46, no. 1 (1998), pp. 1–8.
68. Nemhauser, G.L. and Wolsey, L.A., *Integer and Combinatorial Optimization*, John Wiley & Sons, Inc., New York (1988).
69. Padberg, M.W., Van Roy, T.J., and Wolsey, L.A., “Valid linear inequalities for fixed charge problems,” *Operations Research*, Vol. 33 (1985), pp. 842–861.
70. Papadimitriou, C., *Computational Complexity*, Addison-Wesley, Reading, MA (1994).
71. Papadimitriou, C. and Steiglitz, K., “On the complexity of local search for the traveling salesman problem,” *SIAM Journal on Computing*, Vol. 6 (1977), pp. 76–83.
72. Prais, M. and Ribeiro, C.C., “Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment,” *INFORMS Journal on Computing*, Vol. 12 (2009), pp. 164–176.
73. Rardin, R.L., *Optimization in Operations Research*, Prentice Hall, New Jersey (1998).
74. Reeves, C.R., “Genetic algorithms,” in *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford (1993).
75. Resende, M.G.C., “Greedy randomized adaptive search procedures (GRASP),” in *Encyclopedia of Optimization*, Kluwer Academic Press (1999).
76. Resende, M.G.C. and Ribeiro, C.C., “Greedy randomized adaptive search procedures: advances and applications,” in *Handbook of Metaheuristics*, 2nd edition, J.-Y. Potvin and M. Gendreau (eds), Springer (2010).
77. Roberts, F.S., *Discrete Mathematical Models with Applications to Social, Biological and Environmental Problems*, Prentice Hall, New Jersey (1976).

78. Rushmeier, R. and Kontogiorgis, S., “Advances in the optimization of airline fleet assignment,” *Transportation Science*, Vol. 31, no. 2 (1997), pp. 159–169.
79. Savelsbergh, M.W.P., “Preprocessing and probing techniques for mixed integer programming problems,” *ORSA Journal on Computing*, Vol. 6, no. 4 (1994), pp. 445–454.
80. Schrijver, A., *Theory of Linear and Integer Programming*, John Wiley & Sons, Inc., New York (1986).
81. Sherali, H.D., Lee, Y., and Park, T., “New modeling approaches for the design of local access transport area networks,” *European Journal of Operational Research*, Vol. 127 (2000), pp. 94–108.
82. Toth, P. and Vigo, D. (eds), *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, PA (2002).
83. Trick M.A., “Using sports scheduling to teach integer programming,” *INFORMS Transactions on Education*, Vol. 5, no. 1 (2004), available at <http://ite.pubs.informs.org/Vol5No1/Trick/>.
84. UPS, “Driving success: why the UPS model for managing 103,500 drivers is a competitive advantage,” available at <http://pressroom.ups.com/mdeiakits/popups/factsheet/0,1889,1201,11.html>.
85. Williams, H.P., “Logical problems and integer programming,” *Bulletin of the Institute of Mathematics and its Applications*, Vol. 13, (1977), pp. 18–20.
86. Winston, W.L., *Introduction to Mathematical Programming*, Duxbury Press, California (1995).
87. Winston, W.L., *Operations Research, Applications and Algorithms*, Duxbury Press, California (1993).
88. Winston, W.L. and Albright, S.C., *Practical Management Science*, Duxbury Press, California (1997).
89. Wiper, D.S., Quillinan, J.D., Subramanian, R., Scheff, R.P., Jr., and Marsten, R.E., “Cold-start: fleet assignment at Delta Air Lines,” *Interfaces*, Vol. 24 (1994), pp. 104–120.
90. Wolsey, L.A., *Integer Programming*, John Wiley & Sons, Inc., New York (1998).

91. Yu, G., *Operations Research in the Airline Industry*, Kluwer Academic Publishers, Massachusetts (1998).

INDEX

0-1 Knapsack problem
alternative form
definition
greedy heuristic
Lagrangian dual
Lagrangian relaxation
linear programming relaxation and solution
0-1 Program
Active constraint, tight, binding
Additivity assumption
Adjacent
basic solutions
extreme points
AGIFORS
Aircraft maintenance routing problem
definition
flight cover constraints
infeasibility issues
objective function options
set partitioning model
Airline industry applications
aircraft maintenance routing
crew scheduling
fleet assignment
schedule design
Algorithm
basic simplex method
bounded simplex method
branch-and-bound algorithm
branch-and-cut algorithm

cheapest insertion heuristic for TSP
cycle canceling algorithm
Dijkstra's algorithm
dual simplex method
general cutting plane algorithm
general improving search algorithm
general optimization algorithm
generic augmenting path algorithm
generic cut optimality algorithm
generic label-correcting algorithm
generic local search algorithm
generic shortest path algorithm
genetic algorithm
Gomory's cutting plane algorithm
GRASP
greedy heuristic for 0-1 knapsack problem
heuristic for set covering problem
Kruskal's algorithm
labeling algorithm
minimum cost method
modified label-correcting algorithm
nearest neighbor heuristic for tsp
network simplex method
Prim's algorithm
primal-dual interior point method
simplex method
simulated annealing
tabu search
transportation simplex method
Algorithm design
Alternative (multiple) optimal solution
Applications of linear programming
cabinet manufacturing
candy production
cattle feed blending

engine inventory
equipment replacement
facility location
in oil industry
resource allocation
retail space allocation
sensor placement
shipping
storage
work scheduling

Applications of integer programming
aircraft landing
airline applications
aluminum can labeling
chemical production
college visits
frequency assignment
glass coating
hub allocation
LATA network design
Meals on Wheels
open-pit mining
police station assignment
printed circuit board manufacturing
rental property repair
sports bar television location
sports scheduling
sudoku
telephone cable
waste disposal

Arc
capacities
definition
saturated
Argmax function

Argmin function

Artificial variable

Augmenting path

Balance constraints

Basic (feasible) solution

adjacent

and extreme point

basic variable

definition

degenerate

extended

finite number of

in dual

initial

nonbasic variable

spanning tree as

Basic variable

definition

reduced costs of

Basis

Better formulation

Big-M

Big-M method

computational issues

definition

Bin packing problem

Binary variables

Binding constraint, active, tight

Bland's rule

Blending constraints

Boltzmann distribution

Bounded canonical form

Bounded set

Bounded simplex method
algorithm
extended basic feasible solution
identifying simplex direction
optimality condition
ratio test

Branch-and-bound
active node
algorithm
branch-and-bound tree
branching

- most fractional
- pseudo-costs

strong branching
candidate (incumbent) solution
computational options
fathom(prune) a node
node selection
obtaining bounds
partitioning feasible region
root problem
stopping criteria
use of heuristics

Branch-and-cut
Branching
Branching variable

Candidate solution
Canonical form
bounded
definition

Carathéodory's Theorem
Certainty assumption
Certificate of optimality
Chvátal–Gomory (C–G) inequality

definition and derivation
generated from optimal basic feasible solution
Column generation
computing reduced costs
description
integrality of solution
Complementary slackness
Complementary slackness theorem
Complete enumeration
Concave function
and optimization
definition
linear functions as
Constraints
active, binding, tight
cover
covering
definition
either-or
flight cover
inventory
logical
packing
partitioning
precedence
redundant
shadow price of
subtour elimination
variable bounds
 non-zero
 nonnegativity
Constructive algorithm
basic principles
description
example

Continuous variable
Contour plots
Convex
and linear programs
and optimization
convex combination
convex function
convex hull
convex set
direction of
extreme point of convex set
Corner point
Cover inequality
definition
identifying violated
Covering constraint
CPLEX
Cramer's Rule
Crew assignment problem
Crew pairing problem
Crew scheduling problem
Cut
Cut optimality condition
Cutting plane
Cutting plane algorithm
description
Gomory
algorithm
computational issues
Cutting stock problem
description
linear program
Cycling
and convergence of simplex method

and degeneracy
example
prevention rule (Bland's rule)

Dantzig, G.
Dantzig's rule
Dantzig–Wolfe decomposition
form of linear program
master linear program
phase-I approach
solution approach
Decision variables
Decomposition
Degenerate solution
Demand nodes
Deterministic operations research
Direction
feasible
improving
simplex
unbounded
Discrete optimization problems
Discrete variable
Diversification
Divide and conquer
Divisibility assumption
Dual linear program
and Lagrangian dual
constructing
definition
dual of dual linear program
economic interpretation
Dual simplex method
calculations based on Simplex Method

computational performance

description

interpretation of

Dual variable definition

shadow price as

Duality theorems

complementary slackness

strong duality

weak duality

Edges

Either-or constraints

Entering variable

Enumeration methods

Exact methods

Extended basic (feasible) solution

Extreme direction

Extreme point

adjacency

and basic feasible solution

as potential optimal solutions to linear programs

definition

existence in polyhedron

Facility location models

average distance

fixed-charged location

maximal covering

maximum distance

p -center

p -median

set covering location

Farkas' lemma

and linear programming duality

description

Fathom a node
Feasibility pump
Feasible direction
Feasible flow
Feasible region
definition
relaxing
tighten
Feasible solution
definition
value
Fixed-charge network flow problem
Fixed-charge problem
better formulation
description
Fleet assignment problem
aircraft balance constraint
assumptions
description
Flow
definition
value
value and capacity of s-t cut
Formulation
better
definition
ideal
Fundamental theorem of linear programming
Generalized linear programming
Genetic algorithms crossover
description
fitness function
mutation

population
selecting mating population
solution representation
Global optimal solution
Gomory's cutting plane algorithm
computational performance
description
Graph
acyclic graph
adjacency list
adjacent nodes
arcs
component
connected
cycle
degree of node
directed
edges
forest
incident
indegree of node
induced subgraph
nodes
nontree edges
outdegree of node
path
spanning tree
subgraph
tree
tree edges
undirected
vertices
walk
Graph coloring problem
Greedy randomized adaptive search procedure (GRASP)

and randomization
diversification
improvement phase
performance
restricted candidate list (RCL)
use of memory
vs. greedy methods
Greedy algorithms
0-1 knapsack
definition

Half-space
Heuristic algorithms
for 0-1 knapsack
for set covering
for TSP
Hyperplane
definition
linearly independent
normal vector to

Ideal formulation
convex hull as
definition
Improving direction
definition
in simplex method
of concave functions
of convex functions
Improving search algorithm
calculating step size
convergence of
description
detecting unbounded solutions
finding feasible directions
finding improving directions

Incumbent solution
Infeasible problem
Infinite loop, *see* Loop, infinite
Input parameters
Integer program
Integer programming
combinatorial explosion
formulation
general form
preprocessing
probing
Integer programming (*continued*)
relaxations
solving as linear programs
solving by branch-and-bound
solving by branch-and-cut
solving by cutting plane algorithms
solving via complete enumeration
Integer programming models
facility location
fixed-charge
logical constraints
minimum spanning tree
network design
set covering
traveling salesperson
vehicle routing
Intensification
Interchange methods
2-interchange
2-opt
 k -opt
simplex method as
Interior point methods

Inventory constraints

Job shop scheduling problem

Karush–Kuhn–Tucker (KKT) conditions

Klee–Minty problem

Knapsack constraint cover

cover inequality

extended cover

identify violated cover inequality

minimal cover

Knapsack problem

L_1 -norm

Label-correcting algorithm

convergence

exponential behaviour

FIFO

generic

modified

Labeling algorithm

Lagrange multiplier

Lagrangian dual

and dual linear program

definition

Lagrangian relaxation

Leaving variable

Lifting

definition

importance of order of lifted variables

maximum lifting

using to generate violated valid inequalities

Line search algorithm

Line segment

Linear combination

Linear functions
and concavity
and convexity
Linear program
and dual simplex method
and primal-dual interior point method
and simplex method
block angular form
canonical form
constructing dual problem
definition
degeneracy
dual
Klee–Minty problem
matrix representation
multiple solutions
primal
solving by improving search algorithms
solving in two variables
standard form
various forms
 convert equality constraints into inequalities
 convert inequality constraints into equations
 convert maximization into minimization
 convert nonpositive variables into nonnegative variables

Linear programming
and convexity
assumptions
fundamental theorem
Linear programming models
blending
inventory
linearization of special nonlinear functions
multiperiod
production process

resource allocation

unrestricted values as differences of nonnegative variables

work scheduling

Linear programming relaxation

Linearization of absolute values

Local optima

Local search algorithm

basic principles

definition

example

pros and cons of

simplex method as

Logical constraints

Loop, infinite, *see* Infinite loop

Makespan

Management science

Manhattan distance

Marginal cost, *see* Shadow price

Mathematical model

four-stage cycle

Mathematical program

Matrix

augmented

definition

elementary row operations

equality

identity

inverse

invertible

operations

addition

determinant

multiplication

scalar multiplication

- transpose
- order
- rank
- reduced echelon
- row-echelon
- row equivalence
- square
- totally unimodular
- zero
- Max-flow min-cut theorem
- and linear programming duality
- Maximal covering location problem
- Maximin objective functions
- Maximum distance models
- Maximum flow problem
- assumptions
- definition
- dual formulation of
- generic augmenting path algorithm
- integrality of solution
- labeling algorithm
- linear programming formulation
- max-flow min-cut theorem
- optimality condition
- Metaheuristics
- Miller–Tucker–Zemlin (MTZ) formulation
- Minimax objective function
- Minimum cost method
- Minimum cost network flow problem
- as easy integer program
- assumptions
- basic solution as spanning tree
- definition
- integrality of solution

linear programming formulation
negative cycle optimality condition
Minimum spanning tree problem
algorithms

Kruskal's algorithm

Prim's algorithm

cut optimality condition

definition

formulation size

integer programming formulation

path optimality condition

Mixed-integer program

Modeling languages

AMPL

GAMs

Mosel

MPL

OPL

Multicommodity flow problem

Multiple (alternative) optimal solutions

Negative cycle optimality condition

Neighborhood

Neighborhood search algorithms

description

simplex method as

Network, *see* Graph

Network design problems

Network flow models, *see* Minimum cost network flow problem

Network simplex method

algorithm

and degeneracy

basic solutions as spanning trees

calculating reduced costs

updating basis and solution

Node
active
definition
degree
demand
root
sink
source
supply
transshipment
Node selection
Nonbasic variable
Nonlinear program
general form
optimal solutions
Nontree edges
Normal vector

Objective function
Operations management
Operations research
Optimal solution
at a basic feasible solution
at an extreme point
definition
value
Optimal value
Optimality condition
Optimization algorithm classes
exact methods
heuristic methods
Optimization model
Optimization modeling
introduction
model formulation

Overlapping covers problem

p -center problem

p -median problem

Packing constraint

Parametric programming

Partitioning constraint

Path

Path optimality condition

Piecewise linear function

Polyhedron

basic (feasible) solutions

convex sets

defining hyperplanes

 definition

 linear independence

definition

edge

extreme points

 definition

 existence of

 representation theorem

Polytope

Precedence constraints

Preprocessing

Primal-dual interior point method

comparison to simplex method

convergence

description

Primal linear program

Primal simplex method, *see* Simplex method

Probing

0-1 knapsack problem

using linear programming relaxation

Proportionality assumption

Prune a node

Rank of a matrix

Ratio test

Rectilinear distance

Reduced cost

Reduced cost vector

Redundant constraint

Relaxation

definition

gap

Lagrangian

linear programming

Representation theorem

Residual

capacity

network

Root node

Root problem

s-t cut

capacity

capacity and value of flow

definition

Saturated arc

Sensitivity analysis

adding constraints

applications of

changes in objective coefficient

changes in right-hand side

definition

graphical approach

objective coefficient ranges

reduced costs

removing constraints
right-hand side ranges
shadow prices
Separation problem
Set
complement
definition
difference
empty set
equality of
intersection
null
size of
subset
union
universal
Set covering location problem
Set covering problem
definition
heuristic
Set packing problem
Set partitioning model
Shadow price
Shortest path problem
assumptions
definition
dual problem
linear programming formulation
optimality condition
Simplex direction
Simplex method
algorithm
alternative interpretation
as local search algorithm

Big-M method
Bland's rule to prevent cycling
bounded simplex
comparison to primal-dual interior point method
convergence of
crashing the basis
cycling
Dantzig's rule
determine improving (simplex) directions
determine step size
Devex pricing
failure to terminate
finding feasible directions
finding initial solution
implications of strong duality theorem
network simplex
partial pricing
ratio test
simplex multipliers
steepest-edge pricing
test for unbounded problem
transportation simplex
two-phase method
updating basis
Simplex multipliers
Simulated annealing
description
initial decisions
temperature cooling
termination conditions
Single machine tardiness problem
Single machine weighted completion time problem
Single-pass algorithms
Sink node
Slack

Slack variable
Solution basic
basic feasible
candidate
degenerate
extended basic (feasible)
feasible
global optimal
incumbent
local optimal
multiple optimal
strictly feasible
Source node
Spanning tree
Standard form of linear programs
Step size
Strictly feasible solution
Strong branching
Strong duality theorem
Subtour
Subtour elimination constraints
Sudoku
Supply nodes
Surplus variable

Tabu search
algorithm
aspiration criteria
diversification and intensification
solution attributes
tabu list
tabu tenure
use of memory
Tight constraint, active, binding
Time-space network

Totally unimodular matrices
Transportation problem
assumptions
balanced
basic solution as spanning tree
definition
dual problem
linear programming formulation
rank of constraint matrix
Transportation simplex method
cycle in tableau
degenerate solution
finding initial solution
finding reduced costs
integrality of optimal solution
tabular form
updating solution
Transshipment nodes
Transshipment problem, *see* Minimum cost network flow problem
Traveling salesperson problem (TSP)
asymmetric
cheapest insertion heuristic
definition
difficulty in solving
integer programming formulation
interchange algorithm
Miller–Tucker–Zemlin (MTZ) formulation
nearest neighbor heuristic
relaxation
subtours
Tree
Two-phase method
description
formulating Phase I problem
infeasible solution to problem

Unbounded direction

Unbounded problem

Unbounded set

Unrestricted variable

Valid coloring

Valid inequalities

and C-G inequality

definition

generating

lifting

use to improve formulation

Value of a feasible solution

Variable

artificial

bounds

non-zero

nonnegativity

branching

continuous

discrete

dual

entering

leaving

slack

surplus

unrestricted in sign

Vector

definition

dimension

linear combination

linear independence

null vector

operations

addition

dot product
multiplication
scalar multiplication
transpose
unit vector
zero vector
Vector of reduced costs
Vehicle routing problem (VRP)
definition
integer programming formulation
subtour elimination constraints
Vertices
Weak duality
Weak duality theorem, 328 “What If” scenarios
Xpress-MP