



Computergesteund Probleemoplossen

(B-KUL-H01G1A)

Prof. dr. ir. Georges Gielen

Prof. dr. ir. Stefan Vandewalle

KU Leuven – Universiteit Leuven

H01G1A : Computergesteund Probleemoplossen

omvang

- 10 x 2 u hoorcollege
- 9 x 2,5 u oefensessies (in computerklas)

docenten: Georges Gielen, Stefan Vandewalle

doelstellingen

- inzicht in de wiskundige modellering van ingenieursproblemen
 - herleiden tot typeprobleem (optimalisatie – netwerken en grafen)
- herkennen van eigenschappen van een probleem en keuze van de juiste oplossingsmethode
- praktische vaardigheid m.b.t. implementatie en oplossing met behulp van softwarepakketten (LINGO)
- inzicht in interpretatie en analyse van de resultaten
- kennis van enkele algoritmen (principe en rekенcomplexiteit)

studiemateriaal

- kopieën uit verschillende, vrij beschikbare handleidingen (afdruk via VTK)
- kopieën van in de les getoonde slides (op Toledo)

software

- LINGO www.lindo.com
- GIDEN: users.iems.northwestern.edu/~giden/

examenvorm

- 6/20 : drie gequoteerde oefeningen in oefensessies
- 14/20 : gesloten-boek examen
- 2 vragen mondeling toe te lichten
- 3 vragen schriftelijk af te geven

H01G1A : Computergesteund Probleemoplossen

Inhoud

1. inleiding (0.5 les)

- toepassingen, classificatie, situering, documentatie

2. lineaire programmatie (LP) (2.5 lessen)

- voorbeelden, basisprincipes, software
- LP-modellering van typische problemen
- algoritmes (simplex, interior point), sensitiviteit

3. grafenalgoritmen en netwerkmodellen (2 lessen)

- kritieke pad analyse, transport, toewijzing
- kortste pad, maximale stroming, minimale opspannende boom, grafenkleuring,...
- oplossing via LP of met gespecialiseerde grafenalgoritmes

4. geheeltallige programmatie (IP) (2 lessen)

- groeperings- en toewijzingsproblemen
- bijzondere problemen (logische condities, niet-lineariteiten,...)
- branch en bound methode

5. speltheorie (1 les)

- constante-som spelen met en zonder zadelpunt
- niet-constante-som spelen

6. niet-lineaire programmatie (1 les)

- theorie: KKT-voorwaarden; praktijk: penalisering- en barrièremethoden
- optimalisatie met meerdere objectieven

7. bijzondere technieken (1 les)

- dynamische programmatie
- globale optimalisatie: simulated annealing, genetische algoritmen

Optimization Modeling with **LINGO**

Sixth Edition

LINDO Systems, Inc. 
1415 North Dayton Street, Chicago, Illinois 60622
Phone: (312)988-7422 Fax: (312)988-9065
E-mail: info@lindo.com

TRADEMARKS

What'sBest! and LINDO are registered trademarks and LINGO is a trademark of LINDO Systems, Inc. Other product and company names mentioned herein are the property of their respective owners.

Copyright © 2015 by LINDO Systems Inc
All rights reserved. First edition 1998
Sixth edition, April 2006
Printed in the United States of America
First printing 2003
This printing January 2015

ISBN: 1-893355-00-4

Published by



LINDO SYSTEMS INC.

1415 North Dayton Street
Chicago, Illinois 60622
Technical Support: (312) 988-9421
<http://www.lindo.com>
e-mail: tech@lindo.com

Contents

Contents	iii
Preface	xiii
Acknowledgments.....	xiii
1.....	1
What Is Optimization?.....	1
1.1 Introduction	1
1.2 A Simple Product Mix Problem	1
1.2.1 Graphical Analysis	2
1.3 Linearity	5
1.4 Analysis of LP Solutions	6
1.5 Sensitivity Analysis, Reduced Costs, and Dual Prices	8
1.5.1 Reduced Costs	8
1.5.2 Dual Prices.....	8
1.6 Unbounded Formulations	9
1.7 Infeasible Formulations	10
1.8 Multiple Optimal Solutions and Degeneracy	11
1.8.1 The “Snake Eyes” Condition	13
1.8.2 Degeneracy and Redundant Constraints.....	15
1.9 Nonlinear Models and Global Optimization	16
1.10 Problems.....	18
2.....	21
Solving Math Programs with LINGO	21
2.1 Introduction	21
2.2 LINGO for Windows and Apple Mac.....	21
2.2.3 LINGO Menu	23
2.2.4 Windows Menu	24
2.2.5 Help Menu.....	24
2.2.6 Summary.....	25
2.3 Getting Started on a Small Problem	25
2.4 Integer Programming with LINGO	26
2.4.1 Warning for Integer Programs	28
2.5 Solving an Optimization Model	28
2.6 Problems.....	29
3.....	31
Analyzing Solutions	31
3.1 Economic Analysis of Solution Reports.....	31
3.2 Economic Relationship Between Dual Prices and Reduced Costs.....	31
3.2.1 The Costing Out Operation: An Illustration	32
3.2.2 Dual Prices, LaGrange Multipliers, KKT Conditions, and Activity Costing	33
3.3 Range of Validity of Reduced Costs and Dual Prices	34
3.3.1 Predicting the Effect of Simultaneous Changes in Parameters—The 100% Rule .	39
3.4 Sensitivity Analysis of the Constraint Coefficients.....	40
3.5 The Dual LP Problem, or the Landlord and the Renter	41

3.6 Problems.....	43
4.....	49
The Model Formulation Process.....	49
4.1 The Overall Process	49
4.2 Approaches to Model Formulation.....	50
4.3 The Template Approach	50
4.3.1 Product Mix Problems	50
4.3.2 Covering, Staffing, and Cutting Stock Problems.....	50
4.3.3 Blending Problems.....	50
4.3.4 Multiperiod Planning Problems	51
4.3.5 Network, Distribution, and PERT/CPM Models	51
4.3.6 Multiperiod Planning Problems with Random Elements.....	51
4.3.7 Financial Portfolio Models	51
4.3.8 Game Theory Models	52
4.4 Constructive Approach to Model Formulation	52
4.4.1 Example	53
4.4.2 Formulating Our Example Problem	53
4.5 Choosing Costs Correctly	54
4.5.1 Sunk vs. Variable Costs.....	54
4.5.2 Joint Products	56
4.5.3 Book Value vs. Market Value.....	57
4.6 Common Errors in Formulating Models.....	59
4.7 The Nonsimultaneity Error.....	61
4.8 Debugging a Model	61
4.9 Problems.....	63
5.....	67
The Sets View of the World	67
5.1 Introduction	67
5.1.1 Why Use Sets?	67
5.1.2 What Are Sets?	67
5.1.3 Types of Sets	68
5.2 The SETS Section of a Model	68
5.2.1 Defining Primitive Sets	68
5.2.2 Defining Derived Sets	69
5.2.3 Summary.....	70
5.3 The DATA Section	71
5.4 Set Looping Functions	73
5.4.1 @SUM Set Looping Function	74
5.4.2 @MIN and @MAX Set Looping Functions	75
5.4.3 @FOR Set Looping Function	76
5.4.4 Nested Set Looping Functions	77
5.5 Set Based Modeling Examples.....	77
5.5.1 Primitive Set Example	78
5.5.2 Dense Derived Set Example.....	81
5.5.3 Sparse Derived Set Example - Explicit List	83
5.5.4 A Sparse Derived Set Using a Membership Filter	88
5.6 Domain Functions for Variables	92

5.7 Spreadsheets and LINGO	92
5.8 Programming in LINGO	96
5.8.1 Building Blocks for Programming.....	96
5.8.2 Generating Graphs and Charts.....	98
5.9 Problems.....	100
6.....	101
Product Mix Problems	101
6.1 Introduction	101
6.2 Example.....	102
6.3 Process Selection Product Mix Problems	105
6.4 Problems.....	110
7.....	113
Covering, Staffing & Cutting Stock Models.....	113
7.1 Introduction	113
7.1.1 Staffing Problems.....	114
7.1.2 Example: Northeast Tollway Staffing Problems.....	114
7.1.3 Additional Staff Scheduling Features.....	116
7.2 Cutting Stock and Pattern Selection.....	117
7.2.1 Example: Cooldot Cutting Stock Problem.....	118
7.2.2 Formulation and Solution of Cooldot	119
7.2.3 Generalizations of the Cutting Stock Problem.....	123
7.2.4 Two-Dimensional Cutting Stock Problems	125
7.3 Crew Scheduling Problems	126
7.3.1 Example: Sayre-Priors Crew Scheduling.....	126
7.3.2 Solving the Sayre/Priors Crew Scheduling Problem	128
7.3.3 Additional Practical Details	131
7.4 A Generic Covering/Partitioning/Packing Model	132
7.5 Problems.....	134
8.....	145
Networks, Distribution and PERT/CPM.....	145
8.1 What's Special About Network Models.....	145
8.1.1 Special Cases	148
8.2 PERT/CPM Networks and LP	148
8.3 Activity-on-Arc vs. Activity-on-Node Network Diagrams.....	153
8.4 Crashing of Project Networks	154
8.4.1 The Cost and Value of Crashing.....	155
8.4.2 The Cost of Crashing an Activity	155
8.4.3 The Value of Crashing a Project.....	155
8.4.4 Formulation of the Crashing Problem	156
8.5 Resource Constraints in Project Scheduling	159
8.6 Path Formulations.....	161
8.6.1 Example	162
8.7 Path Formulations of Undirected Networks	163
8.7.1 Example	164
8.8 Double Entry Bookkeeping: A Network Model of the Firm	166
8.9 Extensions of Network LP Models.....	167
8.9.1 Multicommodity Network Flows	168

8.9.2 Reducing the Size of Multicommodity Problems	169
8.9.3 Multicommodity Flow Example	169
8.9.4 Fleet Routing and Assignment.....	172
8.9.5 Fleet Assignment	175
8.9.6 Leontief Flow Models	179
8.9.7 Activity/Resource Diagrams.....	183
8.9.8 Spanning Trees.....	185
8.9.9 Steiner Trees	187
8.10 Nonlinear Networks	191
8.11 Problems.....	194
9.....	203
Multi-period Planning Problems	203
9.1 Introduction	203
9.2 A Dynamic Production Problem.....	205
9.2.1 Formulation	205
9.2.2 Constraints	206
9.2.3 Representing Absolute Values.....	208
9.3 Multi-period Financial Models	209
9.3.1 Example: Cash Flow Matching	209
9.4 Financial Planning Models with Tax Considerations	213
9.4.1 Formulation and Solution of the WSDM Problem	214
9.4.2 Interpretation of the Dual Prices	215
9.5 Present Value vs. LP Analysis.....	216
9.6 Accounting for Income Taxes	217
9.7 Dynamic or Multi-period Networks.....	220
9.8 End Effects	222
9.8.1 Perishability/Shelf Life Constraints	223
9.8.2 Startup and Shutdown Costs	223
9.9 Non-optimality of Cyclic Solutions to Cyclic Problems	223
9.10 Problems.....	229
10.....	233
Blending of Input Materials	233
10.1 Introduction	233
10.2 The Structure of Blending Problems.....	234
10.2.1 Example: The Pittsburgh Steel Company Blending Problem	235
10.2.2 Formulation and Solution of the Pittsburgh Steel Blending Problem.....	236
10.3 A Blending Problem within a Product Mix Problem	238
10.3.1 Formulation	239
10.3.2 Representing Two-sided Constraints.....	240
10.4 Proper Choice of Alternate Interpretations of Quality Requirements	243
10.5 How to Compute Blended Quality	246
10.5.1 Example	246
10.5.2 Generalized Mean.....	247
10.6 Interpretation of Dual Prices for Blending Constraints	248
10.7 Fractional or Hyperbolic Programming	249
10.8 Multi-Level Blending: Pooling Problems	250
10.9 Problems.....	255

11.....	267
Formulating and Solving Integer Programs	267
11.1 Introduction	267
11.1.1 Types of Variables	267
11.2 Exploiting the IP Capability: Standard Applications.....	268
11.2.1 Binary Representation of General Integer Variables	268
11.2.2 Minimum Batch Size Constraints	268
11.2.3 Fixed Charge Problems	269
11.2.4 The Simple Plant Location Problem	269
11.2.5 The Capacitated Plant Location Problem (CPL).....	271
11.2.6 Modeling Alternatives with the Scenario Approach	273
11.2.7 Linearizing a Piecewise Linear Function, Discontinuous Case	274
11.2.10 Converting Multivariate Functions to Separable Functions	280
11.3 Outline of Integer Programming Methods	281
11.4 Computational Difficulty of Integer Programs.....	284
11.4.1 NP-Complete Problems	285
11.5 Problems with Naturally Integer Solutions and the Prayer Algorithm.....	286
11.5.1 Network LPs Revisited	286
11.5.2 Integral Leontief Constraints	286
11.5.3 Example: A One-Period MRP Problem.....	287
11.5.4 Transformations to Naturally Integer Formulations	289
11.6 The Assignment Problem and Related Sequencing and Routing Problems.....	291
11.6.1 Example: The Assignment Problem	291
11.6.2 The Traveling Salesperson Problem	293
11.6.3 Capacitated Multiple TSP/Vehicle Routing Problems.....	300
11.6.4 Minimum Spanning Tree.....	303
11.6.5 The Linear Ordering Problem	304
11.6.6 Quadratic Assignment Problem	306
11.7 Problems of Grouping, Matching, Covering, Partitioning, and Packing	310
11.7.1 Formulation as an Assignment Problem.....	311
11.7.2 Matching Problems, Groups of Size Two	311
11.7.3 Groups with More Than Two Members	314
11.7.4 Groups with a Variable Number of Members, Assignment Version	317
11.7.5 Groups with A Variable Number of Members, Packing Version	318
11.7.6 Groups with A Variable Number of Members, Cutting Stock Problem	321
11.7.7 Groups with A Variable Number of Members, Vehicle Routing.....	325
11.8 Linearizing Products of Variables	329
11.8.1 Example: Bundling of Products.....	330
11.9 Representing Logical Conditions.....	332
11.10 Problems.....	333
12.....	343
Decision making Under Uncertainty and Stochastic Programs	343
12.1 Introduction	343
12.1.1 Identifying Sources of Uncertainty.....	344
12.2 Formulation and Structure of an SP Problem.....	345
12.3 Single Stage Decisions Under Uncertainty	346
12.3.1 The News Vendor Problem.....	346

12.3.2 Multi-product Inventory with Repositioning	349
12.x Multi-Stage Decisions Under Uncertainty	352
12.x.1 Stopping Rule and Option to Exercise Problems	353
12.3 The Scenario Approach	357
12.5 Expected Value of Perfect Information (EVPI)	357
12.6 Expected Value of Modeling Uncertainty	358
12.6.1 Certainty Equivalence	358
12.7 Risk Aversion	359
12.7.1 Downside Risk	360
12.7.2 Example	361
12.9.1 Dynamic Programming and Financial Option Models	364
12.9.2 Binomial Tree Models of Interest Rates	365
12.9.3 Binomial Tree Models of Foreign Exchange Rates	369
12.10 Decisions Under Uncertainty with an Infinite Number of Periods	371
12.10.1 Example: Cash Balance Management	373
12.11 Chance-Constrained Programs	376
12.12 Problems	377
13.....	379
Portfolio Optimization.....	379
13.1 Introduction	379
13.2 The Markowitz Mean/Variance Portfolio Model	379
13.2.1 Example	380
13.3 Dualing Objectives: Efficient Frontier and Parametric Analysis	383
13.3.1 Portfolios with a Risk-Free Asset	383
13.3.2 The Sharpe Ratio	386
13.4 Important Variations of the Portfolio Model	387
13.4.1 Portfolios with Transaction Costs	388
13.4.2 Example	388
13.4.3 Portfolios with Taxes	390
13.4.4 Factors Model for Simplifying the Covariance Structure	392
13.4.5 Example of the Factor Model	393
13.4.6 Scenario Model for Representing Uncertainty	394
13.4.7 Example: Scenario Model for Representing Uncertainty	395
13.5 Measures of Risk other than Variance	397
13.5.1 Value at Risk(VaR)	398
13.5.2 Example of VaR	398
13.5.3 VaR Anomalies	400
13.5.4 Conditional Value at Risk(CVaR)	401
13.6 Scenario Model and Minimizing Downside Risk	403
13.6.1 Semi-variance and Downside Risk	404
13.6.2 Downside Risk and MAD	406
13.6.3 Scenarios Based Directly Upon a Covariance Matrix	406
13.7 Hedging, Matching and Program Trading	408
13.7.1 Portfolio Hedging	408
13.7.2 Portfolio Matching, Tracking, and Program Trading	408
13.8 Methods for Constructing Benchmark Portfolios	409
13.8.1 Scenario Approach to Benchmark Portfolios	412

13.8.2 Efficient Benchmark Portfolios	414
13.8.3 Efficient Formulation of Portfolio Problems.....	415
13.9 Cholesky Factorization for Quadratic Programs.....	417
13.10 Problems.....	419
14.....	421
Multiple Criteria and Goal Programming.....	421
14.1 Introduction	421
14.1.1 Alternate Optima and Multicriteria	422
14.2 Approaches to Multi-criteria Problems	422
14.2.1 Pareto Optimal Solutions and Multiple Criteria	422
14.2.2 Utility Function Approach	422
14.2.3 Trade-off Curves	423
14.2.4 Example: Ad Lib Marketing	423
14.3 Goal Programming and Soft Constraints.....	426
14.3.1 Example: Secondary Criterion to Choose Among Alternate Optima	427
14.3.2 Preemptive/Lexico Goal Programming	429
14.4 Minimizing the Maximum Hurt, or Unordered Lexico Minimization	432
14.4.1 Example	433
14.4.2 Finding a Unique Solution Minimizing the Maximum	433
14.5 Identifying Points on the Efficient Frontier	438
14.5.1 Efficient Points, More-is-Better Case	438
14.5.2 Efficient Points, Less-is-Better Case	440
14.5.3 Efficient Points, the Mixed Case	442
14.6 Comparing Performance with Data Envelopment Analysis.....	443
14.7 Problems.....	448
15.....	451
Economic Equilibria and Pricing	451
15.1 What is an Equilibrium?	451
15.2 A Simple Simultaneous Price/Production Decision	452
15.3 Representing Supply & Demand Curves in LPs.....	453
15.4 Auctions as Economic Equilibria	457
15.5 Multi-Product Pricing Problems	461
15.6 General Equilibrium Models of An Economy.....	465
15.7 Transportation Equilibria.....	467
15.7.1 User Equilibrium vs. Social Optimum	471
15.8 Equilibria in Networks as Optimization Problems.....	473
15.8.1 Equilibrium Network Flows.....	475
15.9 Problems.....	477
16.....	481
Game Theory and Cost Allocation	481
16.1 Introduction	481
16.2 Two-Person Games.....	481
16.2.1 The Minimax Strategy	482
16.3 Two-Person Non-Constant Sum Games	484
16.3.1 Prisoner's Dilemma.....	485
16.3.2 Choosing a Strategy	486
16.3.3 Bimatrix Games with Several Solutions	489

1

What Is Optimization?

1.1 Introduction

Optimization, or constrained optimization, or mathematical programming, is a mathematical procedure for determining optimal allocation of scarce resources. Optimization, and its most popular special form, Linear Programming (LP), has found practical application in almost all facets of business, from advertising to production planning. Transportation and aggregate production planning problems are the most typical objects of LP analysis. The petroleum industry was an early intensive user of LP for solving fuel blending problems.

It is important for the reader to appreciate at the outset that the “programming” in Mathematical Programming is of a different flavor than the “programming” in Computer Programming. In the former case, it means to plan and organize (as in “Get with the program!”). In the latter case, it means to write instructions for performing calculations. Although aptitude in one suggests aptitude in the other, training in the one kind of programming has very little direct relevance to the other.

For most optimization problems, one can think of there being *two important classes of objects*. The first of these is *limited resources*, such as land, plant capacity, and sales force size. The second is *activities*, such as “produce low carbon steel,” “produce stainless steel,” and “produce high carbon steel.” *Each activity consumes* or possibly *contributes* additional amounts of the *resources*. The problem is to determine the best combination of activity levels that does not use more resources than are actually available. We can best gain the flavor of LP by using a simple example.

1.2 A Simple Product Mix Problem

The Enginola Television Company produces two types of TV sets, the “Astro” and the “Cosmo”. There are two production lines, one for each set. The Astro production line has a capacity of 60 sets per day, whereas the capacity for the Cosmo production line is only 50 sets per day. The labor requirements for the Astro set is 1 person-hour, whereas the Cosmo requires a full 2 person-hours of labor. Presently, there is a maximum of 120 man-hours of labor per day that can be assigned to production of the two types of sets. If the profit contributions are \$20 and \$30 for each Astro and Cosmo set, respectively, what should be the daily production?

2 Chapter 1 What is Optimization?

A structured, but verbal, description of what we want to do is:

Maximize Profit contribution
subject to Astro production less-than-or-equal-to Astro capacity,
Cosmo production less-than-or-equal-to Cosmo capacity,
Labor used less-than-or-equal-to labor availability.

Until there is a significant improvement in artificial intelligence/expert system software, we will need to be more precise if we wish to get some help in solving our problem. We can be more precise if we define:

A = units of Astros to be produced per day,
 C = units of Cosmos to be produced per day.

Further, we decide to measure:

Profit contribution in dollars,
Astro usage in units of Astros produced,
Cosmo usage in units of Cosmos produced, and
Labor in person-hours.

Then, a precise statement of our problem is:

Maximize $20A + 30C$ (Dollars)
subject to $A \leq 60$ (Astro capacity)
 $C \leq 50$ (Cosmo capacity)
 $A + 2C \leq 120$ (Labor in person-hours)

The first line, “Maximize $20A+30C$ ”, is known as the *objective function*. The remaining three lines are known as *constraints*. Most optimization programs, sometimes called “solvers”, assume all variables are constrained to be nonnegative, so stating the constraints $A \geq 0$ and $C \geq 0$ is unnecessary.

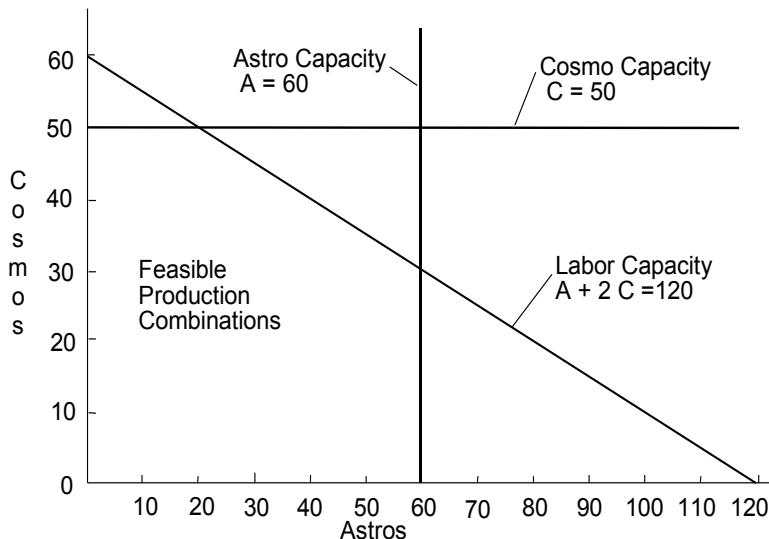
Using the terminology of resources and activities, there are three resources: Astro capacity, Cosmo capacity, and labor capacity. The activities are Astro and Cosmo production. It is generally true that, with each constraint in an optimization model, one can associate some resource. For each decision variable, there is frequently a corresponding physical activity.

1.2.1 Graphical Analysis

The Enginola problem is represented graphically in Figure 1.1. The feasible production combinations are the points in the lower left enclosed by the five solid lines. We want to find the point in the feasible region that gives the highest profit.

To gain some idea of where the maximum profit point lies, let's consider some possibilities. The point $A = C = 0$ is feasible, but it does not help us out much with respect to profits. If we spoke with the manager of the Cosmo line, the response might be: “The Cosmo is our more profitable product. Therefore, we should make as many of it as possible, namely 50, and be satisfied with the profit contribution of $30 \times 50 = \$1500$.”

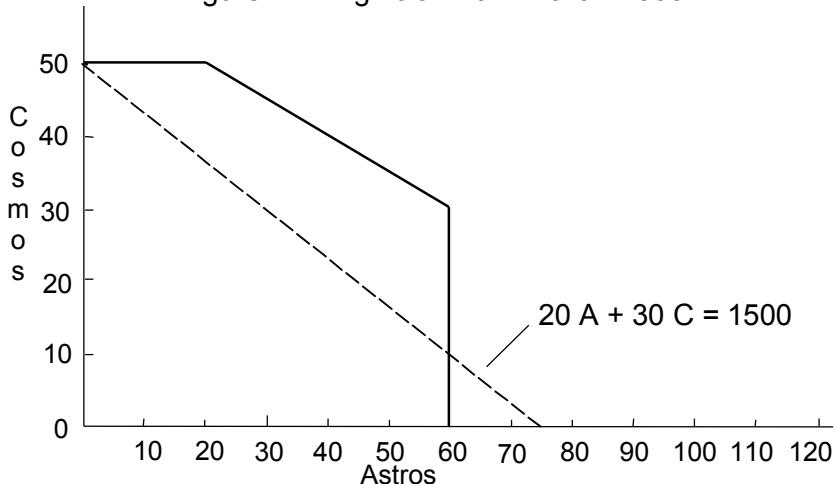
Figure 1.1 Feasible Region for Enginola



You, the thoughtful reader, might observe there are many combinations of A and C , other than just $A = 0$ and $C = 50$, that achieve \$1500 of profit. Indeed, if you plot the line $20A + 30C = 1500$ and add it to the graph, then you get Figure 1.2. Any point on the dotted line segment achieves a profit of \$1500. Any line of constant profit such as that is called an iso-profit line (or iso-cost in the case of a cost minimization problem).

If we next talk with the manager of the Astro line, the response might be: "If you produce 50 Cosmos, you still have enough labor to produce 20 Astros. This would give a profit of $30 \times 50 + 20 \times 20 = \1900 . That is certainly a respectable profit. Why don't we call it a day and go home?"

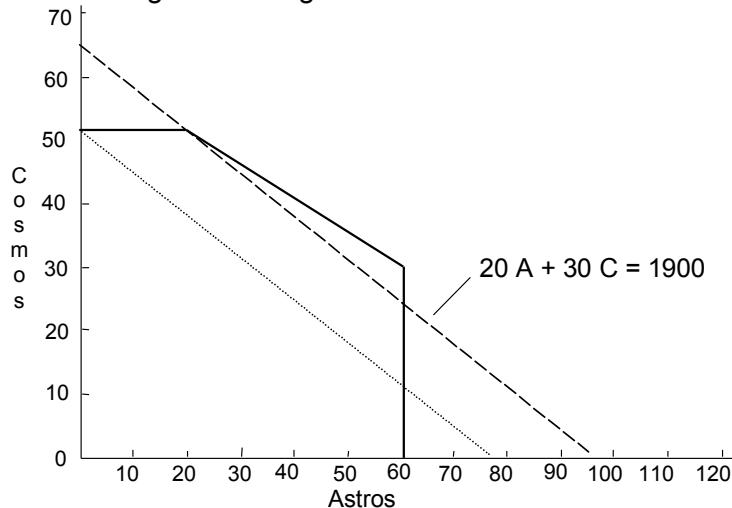
Figure 1.2 Enginola With "Profit = 1500"



4 Chapter 1 What is Optimization?

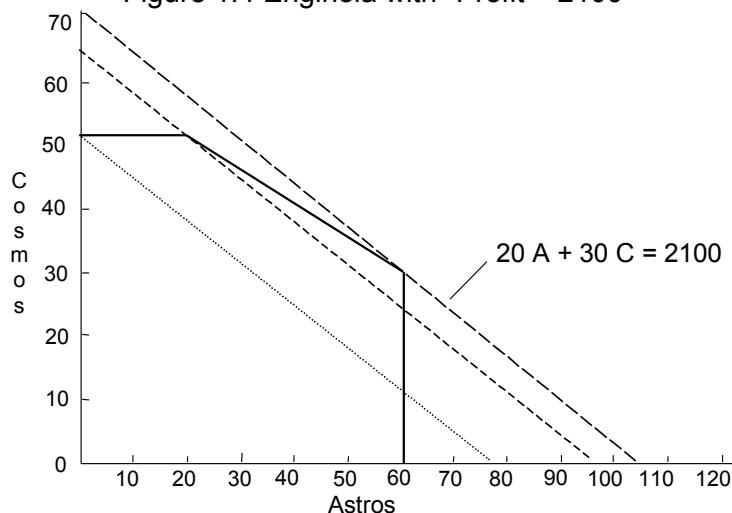
Our ever-alert reader might again observe that there are many ways of making \$1900 of profit. If you plot the line $20A + 30C = 1900$ and add it to the graph, then you get Figure 1.3. Any point on the higher rightmost dotted line segment achieves a profit of \$1900.

Figure 1.3 Enginola with "Profit = 1900"



Now, our ever-perceptive reader makes a leap of insight. As we increase our profit aspirations, the dotted line representing all points that achieve a given profit simply shifts in a parallel fashion. Why not shift it as far as possible for as long as the line contains a feasible point? This last and best feasible point is $A = 60$, $C = 30$. It lies on the line $20A + 30C = 2100$. This is illustrated in Figure 1.4. Notice, even though the profit contribution per unit is higher for Cosmo, we did not make as many (30) as we feasibly could have made (50). Intuitively, this is an optimal solution and, in fact, it is. The graphical analysis of this small problem helps understand what is going on when we analyze larger problems.

Figure 1.4 Enginola with "Profit = 2100"



1.3 Linearity

We have now seen one example. We will return to it regularly. This is an example of a linear mathematical program, or LP for short. Solving linear programs tends to be substantially easier than solving more general mathematical programs. Therefore, it is worthwhile to dwell for a bit on the linearity feature.

Linear programming applies *directly* only to situations in which the effects of the different activities in which we can engage are linear. For practical purposes, we can think of the linearity requirement as consisting of three features:

1. *Proportionality.* The effects of a single variable or activity by itself are proportional (e.g., doubling the amount of steel purchased will double the dollar cost of steel purchased).
2. *Additivity.* The interactions among variables must be additive (e.g., the dollar amount of sales is the sum of the steel dollar sales, the aluminum dollar sales, etc.; whereas the amount of electricity used is the sum of that used to produce steel, aluminum, etc.).
3. *Continuity.* The variables must be continuous (i.e., fractional values for the decision variables, such as 6.38, must be allowed). If both 2 and 3 are feasible values for a variable, then so is 2.51.

A model that includes the two decision variables “price per unit sold” and “quantity of units sold” is probably not linear. The proportionality requirement is satisfied. However, the interaction between the two decision variables is multiplicative rather than additive (i.e., $\text{dollar sales} = \text{price} \times \text{quantity}$, not $\text{price} + \text{quantity}$).

If a supplier gives you quantity discounts on your purchases, then the cost of purchases will not satisfy the proportionality requirement (e.g., the total cost of the stainless steel purchased may be less than proportional to the amount purchased).

A model that includes the decision variable “number of floors to build” might satisfy the proportionality and additivity requirements, but violate the continuity conditions. The recommendation to build 6.38 floors might be difficult to implement unless one had a designer who was ingenious with split level designs. Nevertheless, the solution of an LP might recommend such fractional answers.

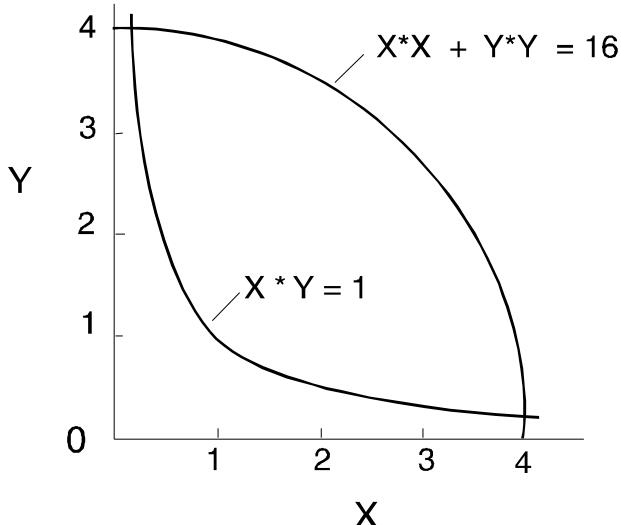
The possible formulations to which LP is applicable are substantially more general than that suggested by the example. The objective function may be minimized rather than maximized; the direction of the constraints may be \geq rather than \leq , or even $=$; and any or all of the parameters (e.g., the 20, 30, 60, 50, 120, 2, or 1) may be negative instead of positive. The principal restriction on the class of problems that can be analyzed results from the linearity restriction.

Fortunately, as we will see later in the chapters on integer programming and quadratic programming, there are other ways of accommodating these violations of linearity.

6 Chapter 1 What is Optimization?

Figure 1.5 illustrates some nonlinear functions. For example, the expression $X \times Y$ satisfies the proportionality requirement, but the effects of X and Y are not additive. In the expression $X^2 + Y^2$, the effects of X and Y are additive, but the effects of each individual variable are not proportional.

Figure 1.5: Nonlinear Relations

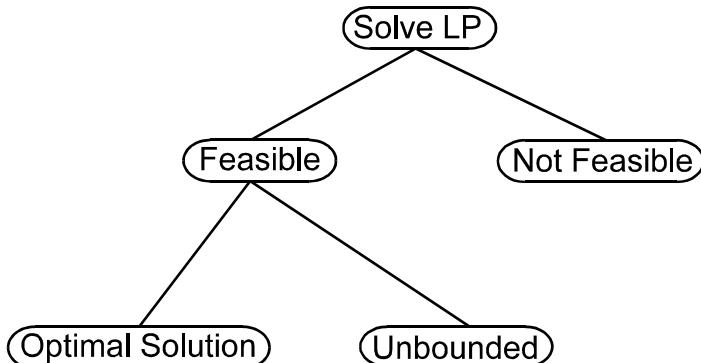


1.4 Analysis of LP Solutions

When you direct the computer to solve a math program, the possible outcomes are indicated in Figure 1.6.

For a properly formulated LP, the leftmost path will be taken. The solution procedure will first attempt to find a feasible solution (i.e., a solution that simultaneously satisfies all constraints, but does not necessarily maximize the objective function). The rightmost, “No Feasible Solution”, path will be taken if the formulator has been too demanding. That is, two or more constraints are specified that cannot be simultaneously satisfied. A simple example is the pair of constraints $x \leq 2$ and $x \geq 3$. The nonexistence of a feasible solution does not depend upon the objective function. It depends solely upon the constraints. In practice, the “No Feasible Solution” outcome might occur in a large complicated problem in which an upper limit was specified on the number of productive hours available and an unrealistically high demand was placed on the number of units to be produced. An alternative message to “No Feasible Solution” is “You Can’t Have Your Cake and Eat It Too”.

Figure 1.6 Solution Outcomes



If a feasible solution has been found, then the procedure attempts to find an optimal solution. If the “Unbounded Solution” termination occurs, it implies the formulation admits the unrealistic result that an infinite amount of profit can be made. A more realistic conclusion is that an important constraint has been omitted or the formulation contains a critical typographical error.

We can solve the Enginola problem in LINGO by typing the following:

```

MODEL:
MAX = 20*A + 30*C;
      A      <= 60;
      C      <= 50;
      A + 2*C <= 120;
END
  
```

We can solve the problem in the Windows version of LINGO by clicking on the red “bullseye” icon. We can get the following solution report by clicking on the “X=” icon”:

Objective value: 2100.000		
Variable	Value	Reduced Cost
A	60.00000	0.00000
C	30.00000	0.00000
Row	Slack or Surplus	Dual Price
1	2100.00000	1.00000
2	0.00000	5.00000
3	20.00000	0.00000
4	0.00000	15.00000

The output has three sections, an informative section, a “variables” section, and a “rows” section. The second two sections are straightforward. The maximum profit solution is to produce 60 Astros and 30 Cosmos for a profit contribution of \$2,100. This solution will leave zero slack in row 2 (the constraint $A \leq 60$), a slack of 20 in row 3 (the constraint $C \leq 50$), and no slack in row 4 (the constraint $A + 2C \leq 120$). Note $60 + 2 \times 30 = 120$.

The third column contains a number of opportunity or marginal cost figures. These are useful by-products of the computations. The interpretation of these “reduced costs” and “dual prices” is discussed in the next section. The reduced cost/dual price section is optional and can be turned on or off by clicking on LINGO | Options | General Solver | Dual Computations | Prices.

1.5 Sensitivity Analysis, Reduced Costs, and Dual Prices

Realistic LPs require large amounts of data. Accurate data are expensive to collect, so we will generally be forced to use data in which we have less than complete confidence. A time-honored adage in data processing circles is “garbage in, garbage out”. A user of a model should be concerned with how the recommendations of the model are altered by changes in the input data. Sensitivity analysis is the term applied to the process of answering this question. Fortunately, an LP solution report provides supplemental information that is useful in sensitivity analysis. This information falls under two headings, reduced costs and dual prices.

Sensitivity analysis can reveal which pieces of information should be estimated most carefully. For example, if it is blatantly obvious that a certain product is unprofitable, then little effort need be expended in accurately estimating its costs. The first law of modeling is "do not waste time accurately estimating a parameter if a modest error in the parameter has little effect on the recommended decision".

1.5.1 Reduced Costs

Associated with each variable in any solution is a quantity known as the *reduced cost*. If the units of the objective function are dollars and the units of the variable are gallons, then the units of the reduced cost are dollars per gallon. The reduced cost of a variable is the amount by which the profit contribution of the variable must be improved (e.g., by reducing its cost) before the variable in question would have a positive value in an optimal solution. Obviously, a variable that already appears in the optimal solution will have a zero reduced cost.

It follows that a second, correct interpretation of the reduced cost is that it is the rate at which the objective function value will deteriorate if a variable, currently at zero, is arbitrarily forced to increase a small amount. Suppose the reduced cost of x is \$2/gallon. This means, if the profitability of x were increased by \$2/gallon, then 1 unit of x (if 1 unit is a “small change”) could be brought into the solution without affecting the total profit. Clearly, the total profit would be reduced by \$2 if x were increased by 1.0 without altering its original profit contribution.

1.5.2 Dual Prices

Associated with each constraint is a quantity known as the *dual price*. If the units of the objective function are cruzeiros and the units of the constraint in question are kilograms, then the units of the dual price are cruzeiros per kilogram. The dual price of a constraint is the rate at which the objective function value will improve as the right-hand side or constant term of the constraint is increased a small amount.

Different optimization programs may use different sign conventions with regard to the dual prices. The LINGO computer program uses the convention that a positive dual price means increasing the right-hand side in question will improve the objective function value. On the other hand, a negative dual price means an increase in the right-hand side will cause the objective function value to deteriorate. A zero dual price means changing the right-hand side a small amount will have no effect on the solution value.

It follows that, under this convention, \leq constraints will have nonnegative dual prices, \geq constraints will have nonpositive dual prices, and $=$ constraints can have dual prices of any sign. Why?

Understanding Dual Prices. It is instructive to analyze the dual prices in the solution to the Enginola problem. The dual price on the constraint $A \leq 60$ is \$5/unit. At first, one might suspect this quantity should be \$20/unit because, if one more Astro is produced, the simple profit contribution of

this unit is \$20. An additional Astro unit will require sacrifices elsewhere, however. Since all of the labor supply is being used, producing more Astros would require the production of Cosmos to be reduced in order to free up labor. The labor tradeoff rate for Astros and Cosmos is $\frac{1}{2}$. That is, producing one more Astro implies reducing Cosmo production by $\frac{1}{2}$ of a unit. The net increase in profits is $\$20 - (1/2) * \$30 = \$5$, because Cosmos have a profit contribution of \$30 per unit.

Now, consider the dual price of \$15/hour on the labor constraint. If we have 1 more hour of labor, it will be used solely to produce more Cosmos. One Cosmo has a profit contribution of \$30/unit. Since 1 hour of labor is only sufficient for one half of a Cosmo, the value of the additional hour of labor is \$15.

1.6 Unbounded Formulations

If we forget to include the labor constraint and the constraint on the production of Cosmos, then an unlimited amount of profit is possible by producing a large number of Cosmos. This is illustrated here:

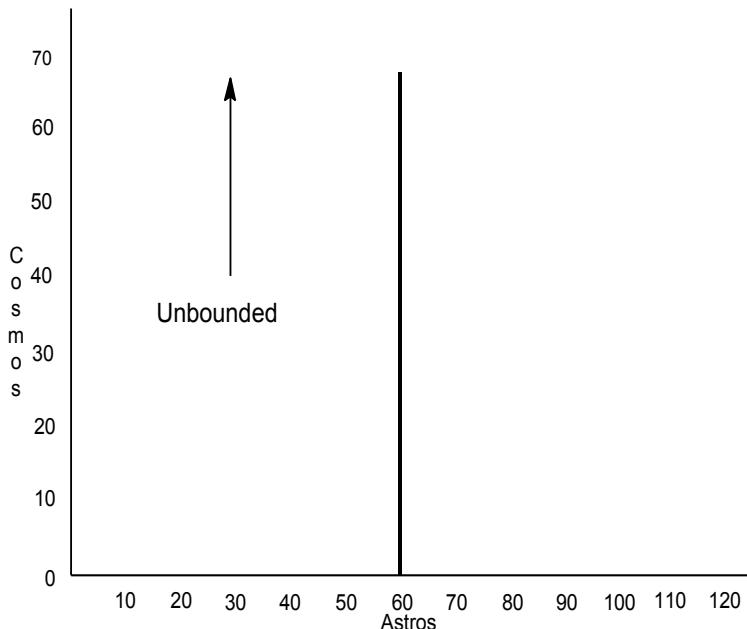
```
MAX = 20 * A + 30 * C;
A <= 60;
```

This generates an error window with the message:

UNBOUNDED SOLUTION

There is nothing to prevent C from being infinitely large. The feasible region is illustrated in Figure 1.7. In larger problems, there are typically several unbounded variables and it is not as easy to identify the manner in which the unboundedness arises.

Figure 1.7 Graph of Unbounded Formulation



1.7 Infeasible Formulations

An example of an infeasible formulation is obtained if the right-hand side of the labor constraint is made 190 and its direction is inadvertently reversed. In this case, the most labor that can be used is to produce 60 Astros and 50 Cosmos for a total labor consumption of $60 + 2 \times 50 = 160$ hours. The formulation and attempted solution are:

```
MAX = (20 * A) + (30 * C);
A <= 60;
C <= 50;
A + 2 * C >= 190;
```

A window with the error message:

NO FEASIBLE SOLUTION.

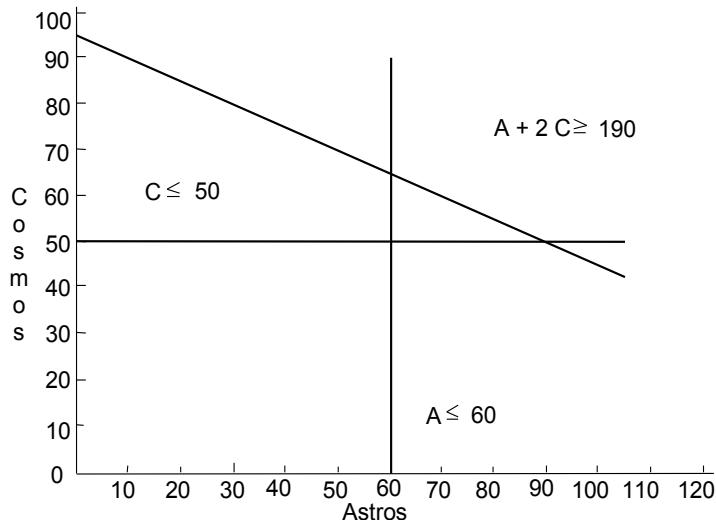
will print. The reports window will generate the following:

Variable	Value	Reduced Cost
A	60.00000	0.0000000
C	50.00000	0.0000000
Row	Slack or Surplus	Dual Price
1	2700.000	0.0000000
2	0.0000000	1.0000000
3	0.0000000	2.0000000
4	-30.00000	-1.0000000

This “solution” is infeasible for the labor constraint by the amount of 30 person-hours ($190 - (1 \times 60 + 2 \times 50)$). The dual prices in this case give information helpful in determining how the infeasibility arose. For example, the +1 associated with row 2 indicates that increasing its right-hand side by one will decrease the infeasibility by 1. The +2 with row 3 means, if we allowed 1 more unit of Cosmo production, the infeasibility would decrease by 2 units because each Cosmo uses 2 hours of labor. The -1 associated with row 4 means that decreasing the right-hand side of the labor constraint by 1 would reduce the infeasibility by 1.

Figure 1.8 illustrates the constraints for this formulation.

Figure 1.8 Graph of Infeasible Formulation



1.8 Multiple Optimal Solutions and Degeneracy

For a given formulation that has a bounded optimal solution, there will be a unique optimum objective function value. However, there may be several different combinations of decision variable values (and associated dual prices) that produce this unique optimal value. Such solutions are said to be degenerate in some sense. In the Enginola problem, for example, suppose the profit contribution of A happened to be \$15 rather than \$20. The problem and a solution are:

```

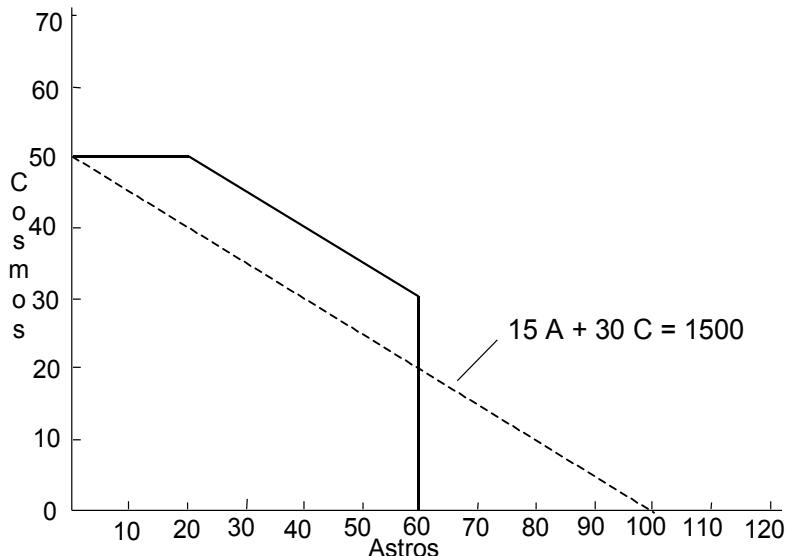
MAX = 15 * A + 30 * C;
A <= 60;
C <= 50;
A + 2 * C <= 120;

Optimal solution found at step: 1
Objective value: 1800.000
Variable      Value      Reduced Cost
    A      20.00000      0.0000000
    C      50.00000      0.0000000

Row    Slack or Surplus    Dual Price
  1      1800.000      1.0000000
  2      40.00000      0.0000000
  3      0.0000000      0.0000000
  4      0.0000000      15.00000

```

Figure 1.9 Model with Alternative Optima



The feasible region, as well as a “profit = 1500” line, are shown in Figure 1.9. Notice the lines $A + 2C = 120$ and $15A + 30C = 1500$ are parallel. It should be apparent that any feasible point on the line $A + 2C = 120$ is optimal.

The particularly observant may have noted in the solution report that the constraint, $C \leq 50$ (i.e., row 3), has both zero slack and a zero dual price. This suggests the production of Cosmos could be decreased a small amount without any effect on total profits. Of course, there would have to be a compensatory increase in the production of Astros. We conclude that there must be an alternate optimum solution that produces more Astros, but fewer Cosmos. We can discover this solution by increasing the profitability of Astros ever so slightly. Observe:

```

MAX = 15.0001 * A + 30 * C;
A <= 60;
C <= 50;
A + 2 * C <= 120;

Optimal solution found at step:           1
Objective value:                      1800.006
Variable          Value      Reduced Cost
      A        60.00000        0.0000000
      C        30.00000        0.0000000
Row    Slack or Surplus      Dual Price
  1        1800.006        1.00000
  2        0.0000000       0.1000000E-03
  3        20.00000        0.0000000
  4        0.0000000       15.00000

```

As predicted, the profit is still about \$1800. However, the production of Cosmos has been decreased to 30 from 50, whereas there has been an increase in the production of Astros to 60 from 20.

1.8.1 The “Snake Eyes” Condition

Alternate optima may exist only if some row in the solution report has zeroes in both the second and third columns of the report, a configuration that some applied statisticians call “snake eyes”. That is, alternate optima may exist only if some variable has both zero value and zero reduced cost, or some constraint has both zero slack and zero dual price. Mathematicians, with no intent of moral judgment, refer to such solutions as degenerate.

If there are alternate optima, you may find your computer gives a different solution from that in the text. However, you should always get the same objective function value.

There are, in fact, two ways in which multiple optimal solutions can occur. For the example in Figure 1.9, the two optimal solution reports differ only in the values of the so-called primal variables (i.e., our original decision variables A, C) and the slack variables in the constraint. There can also be situations where there are multiple optimal solutions in which only the dual variables differ. Consider this variation of the Enginola problem in which the capacity of the Cosmo line has been reduced to 30.

The formulation is:

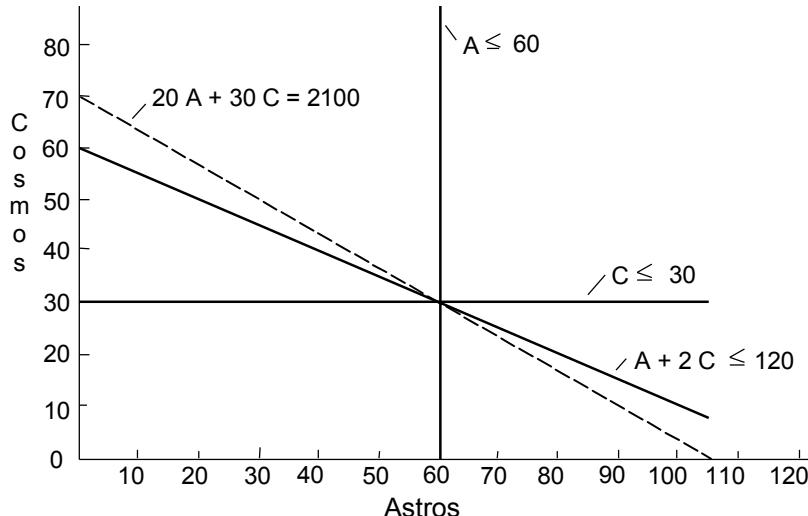
```
MAX = 20 * A + 30 * C;
A < 60;
!note that < and <= are equivalent;
!in LINGO;
C < 30;
A + 2 * C < 120;
```

The corresponding graph of this problem appears in Figure 1.10. An optimal solution is:

Optimal solution found at step:	0	
Objective value:	2100.000	
Variable	Value	Reduced Cost
A	60.00000	0.0000000
C	30.00000	0.0000000
Row	Slack or Surplus	Dual Price
1	2100.000	1.000000
2	0.0000000	20.00000
3	0.0000000	30.00000
4	0.0000000	0.0000000

Again, notice the “snake eyes” in the solution (i.e., the pair of zeroes in a row of the solution report). This suggests the capacity of the Cosmo line (the RHS of row 3) could be changed without changing the objective value. Figure 1.10 illustrates the situation. Three constraints pass through the point $A = 60, C = 30$. Any two of the constraints determine the point. In fact, the constraint $A + 2C \leq 120$ is mathematically redundant (i.e., it could be dropped without changing the feasible region).

Figure 1.10 Alternate Solutions in Dual Variables



If you decrease the RHS of row 3 very slightly, you will get essentially the following solution:

Optimal solution found at step:	0
Objective value:	2100.000
Variable	Value Reduced Cost
A	60.00000 0.0000000
C	30.00000 0.0000000
Row	Slack or Surplus Dual Price
1	2100.000 1.000000
2	0.0000000 5.000000
3	0.0000000 0.0000000
4	0.0000000 15.00000

Notice this solution differs from the previous one only in the dual values.

We can now state the following rule: If a solution report has the “snake eyes” feature (i.e., a pair of zeroes in any row of the report), then there may be an alternate optimal solution that differs either in the primal variables, the dual variables, or in both.

If a solution report exhibits the “snake eyes” configuration, a natural question to ask is: can we determine from the solution report alone whether the alternate optima are in the primal variables or the dual variables? The answer is “no”, as the following two related problems illustrate.

Problem D

$$\begin{aligned} \text{MAX} &= X + Y; \\ &X + Y + Z \leq 1; \\ &X + 2 * Y \leq 1; \end{aligned}$$

Problem P

$$\begin{aligned} \text{MAX} &= X + Y; \\ &X + Y + Z \leq 1; \\ &X + 2 * Z \leq 1; \end{aligned}$$

Both problems possess multiple optimal solutions. The ones that can be identified by the standard simplex solution methods are:

Solution 1

Problem D			Problem P		
OBJECTIVE VALUE			OBJECTIVE VALUE		
1)	Value	Reduced Cost	1)	Value	Reduced Cost
Variable X	1.000000	0.000000	Variable X	1.000000	0.000000
Y	0.000000	0.000000	Y	0.000000	0.000000
Z	0.000000	1.000000	Z	0.000000	1.000000
Row 2)	Slack or Surplus	Dual Prices	Row 2)	Slack or Surplus	Dual Prices
3)	0.000000	1.000000	3)	0.000000	0.000000

Solution 2

Problem D			Problem P		
OBJECTIVE VALUE			OBJECTIVE VALUE		
1)	Value	Reduced Cost	1)	Value	Reduced Cost
Variable X	1.000000	0.000000	Variable X	0.000000	0.000000
Y	0.000000	1.000000	Y	1.000000	0.000000
Z	0.000000	0.000000	Z	0.000000	1.000000
Row 2)	Slack or Surplus	Dual Prices	Row 2)	Slack or Surplus	Dual Prices
3)	0.000000	0.000000	3)	1.000000	0.000000

Notice that:

- *Solution 1* is exactly the same for both problems;
- *Problem D* has multiple optimal solutions in the dual variables (only); while
- *Problem P* has multiple optimal solutions in the primal variables (only).

Thus, one cannot determine from the solution report alone the kind of alternate optima that might exist. You can generate *Solution 1* by setting the RHS of row 3 and the coefficient of *X* in the objective to slightly larger than 1 (e.g., 1.001). Likewise, *Solution 2* is generated by setting the RHS of row 3 and the coefficient of *X* in the objective to slightly less than 1 (e.g., 0.9999).

Some authors refer to a problem that has multiple solutions to the primal variables as *dual degenerate* and a problem with multiple solutions in the dual variables as *primal degenerate*. Other authors say a problem has multiple optima only if there are multiple optimal solutions for the primal variables.

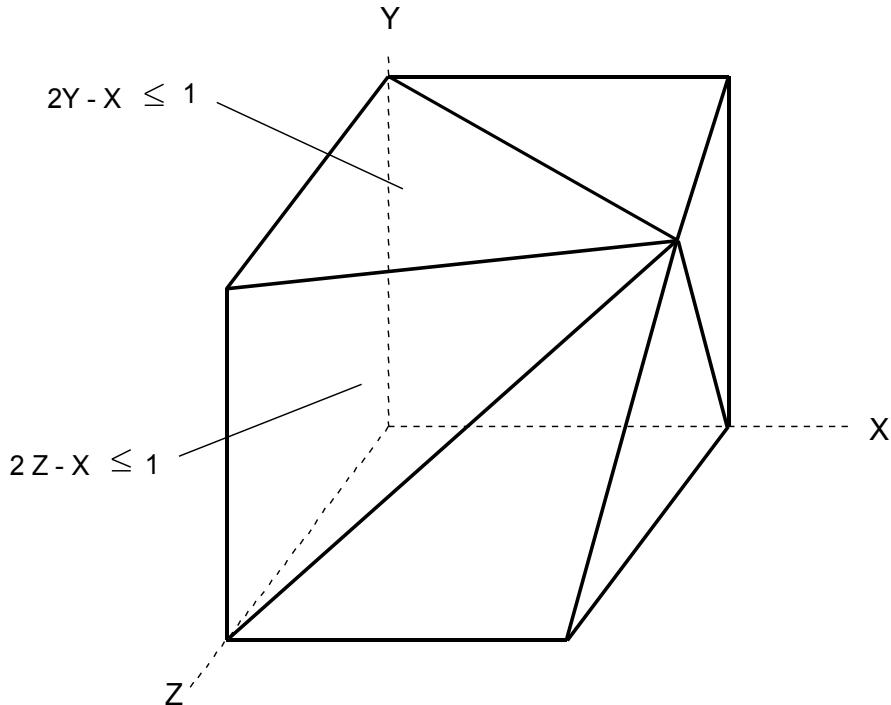
1.8.2 Degeneracy and Redundant Constraints

In small examples, degeneracy usually means there are redundant constraints. In general, however, especially in large problems, degeneracy does not imply there are redundant constraints. The constraint set below and the corresponding Figure 1.11 illustrate:

$$2x - y \leq 1$$

$$\begin{aligned}
 2x - z &\leq 1 \\
 2y - x &\leq 1 \\
 2y - z &\leq 1 \\
 2z - x &\leq 1 \\
 2z - y &\leq 1
 \end{aligned}$$

Figure 1.11 Degeneracy but No Redundancy



These constraints define a cone with apex or point at $x = y = z = 1$, having six sides. The point $x = y = z = 1$ is degenerate because it has more than three constraints passing through it. Nevertheless, none of the constraints are redundant. Notice the point $x = 0.6, y = 0, z = 0.5$ violates the first constraint, but satisfies all the others. Therefore, the first constraint is nonredundant. By trying all six permutations of 0.6, 0, 0.5, you can verify each of the six constraints are nonredundant.

1.9 Nonlinear Models and Global Optimization

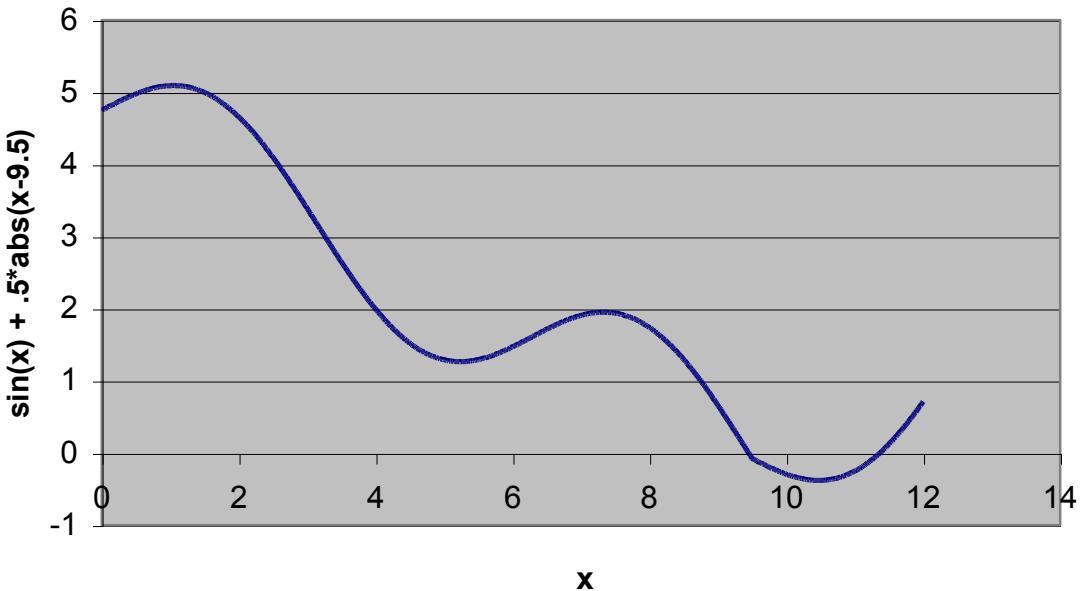
Throughout this text the emphasis is on formulating linear programs. Historically nonlinear models were to be avoided, if possible, for two reasons: a) they take much longer to solve, and b) once “solved” traditional solvers could only guarantee that you had a locally optimal solution. A solution is a local optimum if there is no better solution nearby, although there might be a much better solution some distance away. Traditional nonlinear solvers are like myopic mountain climbers, they can get you to the top of the nearest peak, but they may not see and get you to the highest peak in the mountain range. Versions of LINGO from LINGO 8 onward have a global solver option. If you

check the global solver option, then you are guaranteed to get a global optimum, if you let the solver run long enough. To illustrate, suppose our problem is:

$$\begin{aligned} \text{Min} = & @\sin(x) + .5 * @\text{abs}(x - 9.5); \\ & x \leq 12; \end{aligned}$$

The graph of the function appears in Figure 1.12.

**Figure 1.12 A Nonconvex Function:
 $\sin(x) + .5 * \text{abs}(x - 9.5)$**



If you apply a traditional nonlinear solver to this model you might get one of three solutions: either $x = 0$, or $x = 5.235987$, or $x = 10.47197$. If you check the Global solver option in LINGO, it will report the solution $x = 10.47197$ and label it as a global optimum. Be forewarned that the global solver does not eliminate drawback (a), namely, nonlinear models may take a long time to solve to guaranteed optimality. Nevertheless, the global solver may give a very good, even optimal, solution very quickly but then take a long time to prove that there is no other better solution.

1.10 Problems

1. Your firm produces two products, Thyristors (T) and Lozenges (L), that compete for the scarce resources of your distribution system. For the next planning period, your distribution system has available 6,000 person-hours. Proper distribution of each T requires 3 hours and each L requires 2 hours. The profit contributions per unit are 40 and 30 for T and L , respectively. Product line considerations dictate that at least 1 T must be sold for each 2 L 's.

- (a) Draw the feasible region and draw the profit line that passes through the optimum point.
- (b) By simple common sense arguments, what is the optimal solution?

2. Graph the following LP problem:

$$\begin{aligned} & \text{Minimize } 4X + 6Y \\ & \text{subject to } 5X + 2Y \geq 12 \\ & \quad 3X + 7Y \geq 13 \\ & \quad X \geq 0, Y \geq 0. \end{aligned}$$

In addition, plot the line $4X + 6Y = 18$ and indicate the optimum point.

3. The Volkswagen Company produces two products, the Bug and the SuperBug, which share production facilities. Raw materials costs are \$600 per car for the Bug and \$750 per car for the SuperBug. The Bug requires 4 hours in the foundry/forge area per car; whereas, the SuperBug, because it uses newer more advanced dies, requires only 2 hours in the foundry/forge. The Bug requires 2 hours per car in the assembly plant; whereas, the SuperBug, because it is a more complicated car, requires 3 hours per car in the assembly plant. The available daily capacities in the two areas are 160 hours in the foundry/forge and 180 hours in the assembly plant. Note, if there are multiple machines, the total hours available per day may be greater than 24. The selling price of the Bug at the factory door is \$4800. It is \$5250 for the SuperBug. It is safe to assume whatever number of cars are produced by this factory can be sold.

- (a) Write the linear program formulation of this problem.
- (b) The above description implies the capacities of the two departments (foundry/forge and assembly) are sunk costs. Reformulate the LP under the conditions that each hour of foundry/forge time cost \$90; whereas, each hour of assembly time cost \$60. The capacities remain as before. Unused capacity has no charge.

4. The Keyesport Quarry has two different pits from which it obtains rock. The rock is run through a crusher to produce two products: concrete grade stone and road surface chat. Each ton of rock from the South pit converts into 0.75 tons of stone and 0.25 tons of chat when crushed. Rock from the North pit is of different quality. When it is crushed, it produces a “50-50” split of stone and chat. The Quarry has contracts for 60 tons of stone and 40 tons of chat this planning period. The cost per ton of extracting and crushing rock from the South pit is 1.6 times as costly as from the North pit.
- What are the decision variables in the problem?
 - There are two constraints for this problem. State them in words.
 - Graph the feasible region for this problem.
 - Draw an appropriate objective function line on the graph and indicate graphically and numerically the optimal solution.
 - Suppose all the information given in the problem description is accurate. What additional information might you wish to know before having confidence in this model?
5. A problem faced by railroads is of assembling engine sets for particular trains. There are three important characteristics associated with each engine type, namely, operating cost per hour, horsepower, and tractive power. Associated with each train (e.g., the Super Chief run from Chicago to Los Angeles) is a required horsepower and a required tractive power. The horsepower required depends largely upon the speed required by the run; whereas, the tractive power required depends largely upon the weight of the train and the steepness of the grades encountered on the run. For a particular train, the problem is to find that combination of engines that satisfies the horsepower and tractive power requirements at lowest cost.

In particular, consider the Cimarron Special, the train that runs from Omaha to Santa Fe. This train requires 12,000 horsepower and 50,000 tractive power units. Two engine types, the GM-I and the GM-II, are available for pulling this train. The GM-I has 2,000 horsepower, 10,000 tractive power units, and its variable operating costs are \$150 per hour. The GM-II has 3,000 horsepower, 10,000 tractive power units, and its variable operating costs are \$180 per hour. The engine set may be mixed (e.g., use two GM-I's and three GM-II's).

Write the linear program formulation of this problem.

6. Graph the constraint lines and the objective function line passing through the optimum point and indicate the feasible region for the Enginola problem when:
- All parameters are as given except labor supply is 70 rather than 120.
 - All parameters are as given originally except the variable profit contribution of a Cosmo is \$40 instead of \$30.
7. Consider the problem:

$$\begin{array}{ll} \text{Minimize} & 4x_1 + 3x_2 \\ \text{Subject to} & 2x_1 + x_2 \geq 10 \\ & -3x_1 + 2x_2 \leq 6 \\ & x_1 + x_2 \geq 6 \\ & x_1 \geq 0, x_2 \geq 0 \end{array}$$

Solve the problem graphically.

8. The surgical unit of a small hospital is becoming more concerned about finances. The hospital cannot control or set many of the important factors that determine its financial health. For example, the length of stay in the hospital for a given type of surgery is determined in large part by government regulation. The amount that can be charged for a given type of surgical procedure is controlled largely by the combination of the market and government regulation. Most of the hospital's surgical procedures are elective, so the hospital has considerable control over which patients and associated procedures are attracted and admitted to the hospital. The surgical unit has effectively two scarce resources, the hospital beds available to it (70 in a typical week), and the surgical suite hours available (165 hours in a typical week). Patients admitted to this surgical unit can be classified into the following three categories:

Patient Type	Days of Stay	Surgical Suite Hours Needed	Financial Contribution
A	3	2	\$240
B	5	1.5	\$225
C	6	3	\$425

For example, each type *B* patient admitted will use (i) 5 days of the $7 \times 70 = 490$ bed-days available each week, and (ii) 1.5 hours of the 165 surgical suite hours available each week. One doctor has argued that the surgical unit should try to admit more type *A* patients. Her argument is that, "in terms of \$/days of stay, type *A* is clearly the best, while in terms of \$(/surgical suite hour), it is not much worse than *B* and *C*."

Suppose the surgical unit can in fact control the number of each type of patient admitted each week (i.e., they are decision variables). How many of each type should be admitted each week?

Can you formulate it as an LP?

2

Solving Math Programs with LINGO

2.1 Introduction

The process of solving a math program requires a large number of calculations and is, therefore, best performed by a computer program. The computer program we will use is called LINGO. The main purpose of LINGO is to allow a user to quickly input a model formulation, solve it, assess the correctness or appropriateness of the formulation based on the solution, quickly make minor modifications to the formulation, and repeat the process. LINGO features a wide range of commands, any of which may be invoked at any time. LINGO checks whether a particular command makes sense in a particular context.

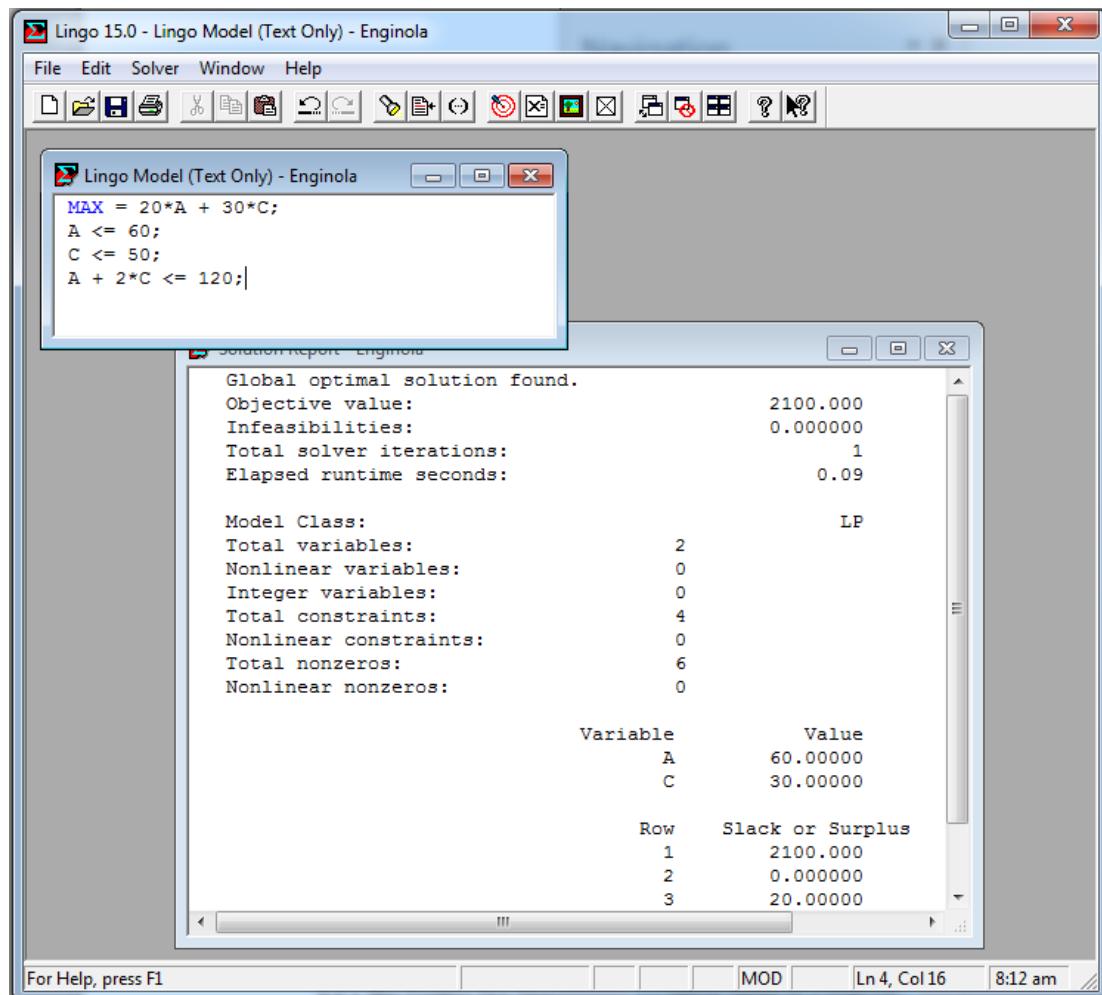
LINGO is available in two variations:

- 1) a version with a graphical user interface (GUI) for Windows and Apple Mac, and
- 2) a generic, text-based version.

The text-based version runs under most popular operating systems, including Unix and Linux. For either version, additional information is available to the user under the Help menu item or Help command. The Windows commands are covered briefly here. After the commands section, we will show you how to enter and solve a simple model.

2.2 LINGO for Windows and Apple Mac

When the GUI version LINGO starts, it opens a blank window known as a *Model Window*. The Model Window is where you “do all your work”. Output in LINGO is displayed in a *Report Window*. LINGO can generate a number of reports pertaining to your model. All the standard commands for opening and saving files, familiar to Windows and Mac users are available. The following is a typical screen shot.



Some of the less common commands available in the GUI version of LINGO are:

SOLVE

Use the *SOLVE* command from the *LINGO/Solver* menu, click on the button, or press *Ctrl+U* to send the model currently in memory to the LINGO solver. If you have more than one model open, the frontmost (or active) window is the one in memory.

MATCH PARENTHESIS Ctrl+P 

Use the *MATCH PARENTHESIS* command from the *Edit* menu, click the button, or type *Ctrl+P* to find the close parenthesis that corresponds to the open parenthesis you have selected.

In addition to this command, there is one other way to find matching parentheses. LINGO will highlight matching parentheses in red when the *Match Paren* option is enabled under the *LINGO|Options* command (see below). By placing the cursor immediately after one of the parentheses of interest, you will notice that the color of the parenthesis changes from black to red. LINGO will simultaneously display the matching parenthesis in red. These parentheses will remain displayed in red until you move the cursor to another position.

PASTE FUNCTION

Use the *PASTE FUNCTION* command from the *Edit* menu to paste any of LINGO's built-in functions at the current insertion point. Choose the category of the LINGO function you want to paste, then select the function from the cascading menu. LINGO inserts place holders for arguments in the functions.

SELECT FONT... Ctrl +J

Use the *SELECT FONT* command from the *Edit* menu or press *Ctrl+J* to select a new font in which to display the currently selected text.

INSERT NEW OBJECT

Use the *INSERT NEW OBJECT* command from the *Edit* menu to embed an OLE object into the LINGO document.

LINKS

Use the *LINKS* command from the *Edit* menu to control the links to external objects in your document.

OBJECT PROPERTIES Alt+Enter

Use the *OBJECT PROPERTIES* command from the *Edit* menu or press *Alt+Enter* to specify the properties of a selected, embedded object

2.2.3 LINGO Menu

SOLUTION... X= Ctrl+W

Use the *SOLUTION* command from the *LINGO* menu, click the button, or press *Ctrl+W* to open the Solutions dialog box. Here you can specify the way you want a report of the solution currently in memory to appear. When you click OK, LINGO writes the report to a Report Window.

GENERATE... Ctrl+G/Ctrl+Q

Use the *DISPLAY MODEL* and *DON'T DISPLAY MODEL* sub-commands from the *LINGO Solver | Generate* command or press *Ctrl+G* or *Ctrl+Q*, respectively, to create an expanded version of the current model. The expanded model explicitly lists all the generated constraints and variables in your model.

If you choose to display the model, LINGO will place a copy of the generated model in a new window, which you may scroll through to examine, print, or save to disk. If you choose not to display the model, LINDO will generate the model without displaying it, but will store the generated model for later use by the appropriate solver.

PICTURE  **Ctrl+K**

Use the *PICTURE* command from the *LINGO* menu or press *Ctrl+K* to display a model in matrix form. Viewing the model in matrix form can be helpful in identifying special structure in your model.

2.2.4 Windows Menu

COMMAND WINDOW **Ctrl +1**

Use the *COMMAND WINDOW* command from the *Windows* menu or press *Ctrl+I* to open LINGO's Command Window. The Command Window gives you access to LINGO's command line interface. In general, Windows users will not need to make use of the Command Window. It is provided for users who may want to put together application-specific "products" that make use of LINGO through Command Window scripts to control the program. Please refer to your help file or user's manual for more information on the command line commands.

STATUS WINDOW **Ctrl +2**

Use the *STATUS WINDOW* command from the *Windows* menu or press *Ctrl+2* to open LINGO's Solver Status window.

2.2.5 Help Menu

HELP TOPICS 

Use the *HELP TOPICS* command from the *Help* menu, or click on the first question mark button to open LINGO help to the Contents section. Press the second button (with the arrow) to invoke context-sensitive help. Once the cursor has changed to the question mark, selecting any command will take you to help for that command.

REGISTER

Use the *REGISTER* command from the *Help* menu to register your version of LINGO online. You will need a connection to the internet open for this command to work. Enter your personal information in the dialog box supplied and select the register button. Your information will be sent directly to LINDO Systems via the Internet.

LINDO Systems is constantly working to make our products faster and easier to use. Registering your software with LINDO ensures that you will be kept up-to-date on the latest enhancements and other product news.

AUTOUPDATE

Use the *AUTOUPDATE* command from the *Help* menu to have LINGO automatically check every time you start the LINGO software whether there is a more recent version of LINGO available for download on the LINDO Systems website. You will need a connection to the internet open for this command to work.

ABOUT LINGO...

Use the *ABOUT LINGO* command from the *Help* menu to view information about the version of LINGO you are currently using (e.g., the release number, constraint limit, variable limit, and memory limit).

2.2.6 Summary

This is not intended to be an exhaustive description of the commands available in the Windows version of LINGO. Please refer to your help file or user's manual for a more in-depth analysis.

2.3 Getting Started on a Small Problem

When you start LINGO for Windows, the program opens an *<untitled>* window for you. For purposes of introduction, let's enter the Enginola problem we looked at in the previous chapter directly into this *<untitled>* window:

```
MAX = (20 * A) + (30 * C);
!note that the parentheses aren't needed, because LINGO;
!will do multiplication and division first;
A < 60;
C < 50;
A + 2 * C < 120;
```

Note, even though the strict inequality, “<”, was entered above, LINGO interprets it as the loose inequality, “≤”. The reason is that typical keyboards have only the strict inequalities, < and >. You may, and in fact are encouraged to, use the two symbols “≤=” to emphasize an inequality is of a less-than-or-equal-to nature. Also, notice comments are preceded by the exclamation mark (!). A semicolon (;) terminates a comment.



Click on the Solve/“bullseye” button , use the *Solve* command from the *Solve* menu, or press *Ctrl+U* to solve the model. While solving, LINGO will show the Solver Status Window with information about the model and the solution process. When it's done solving, the “State” field should read “Global Optimum”. Then, click on the “Close” button to close the Solver Status Window:

The following solution is now in a Report Window:

Optimal solution found at step:	1
Objective value:	2100.000
Variable	Value Reduced Cost
A	60.00000 0.0000000
C	30.00000 0.0000000
Row	Slack or Surplus Dual Price
1	2100.000 1.000000
2	0.0000000 5.000000
3	20.00000 0.0000000
4	0.0000000 15.00000

Editing the model is simply a matter of finding and changing the variable, coefficient, or direction you want to change. Any changes will be taken into account the next time that you solve the model.



Click on the button, use the *Save* command from the *File* menu, or press *Ctrl+S* to save your work.

2.4 Integer Programming with LINGO

Fairly shortly after you start looking at problems for which optimization might be applicable, you discover the need to restrict certain variables to integer values (i.e., 0, 1, 2, etc.). LINGO allows you to identify such variables. We give an introductory treatment here. It is discussed more thoroughly in Chapter 11, *Formulating and Solving Integer Programs*. Integer variables in LINGO can be either 0/1 or general. Variables restricted to the values 0 or 1 are identified with the *@BIN* specification. Variables that may be 0, 1, 2, etc., are identified with the *@GIN* specification.

In the following model, the variables *TOM*, *DICK*, and *HARRY* are restricted to be 0 or 1:

```
MAX = 4 * TOM + 3 * DICK + 2 * HARRY;
      2.5 * TOM + 3.1 * HARRY <= 5;
      .2 * TOM + .7 * DICK + .4 * HARRY <= 1;
@BIN(TOM);
@BIN(DICK);
@BIN(HARRY);
```



After solving, to see the solution, choose *Solution* from the *Reports* menu, or click on the button, and choose *All Values*. The Report Window displays the following:

Optimal solution found at step:	1
Objective value:	7.000000
Branch count:	0
Variable	Value Reduced Cost
TOM	1.000000 -4.000000
DICK	1.000000 -3.000000
HARRY	0.0000000 -2.000000
Row	Slack or Surplus Dual Price
1	7.000000 1.000000
2	2.500000 0.0000000
3	0.1000000 0.0000000

General integers, which can be 0, 1, 2, etc., are identified in analogous fashion by using `@GIN` instead of `@BIN`, for example:

```
@GIN(TONIC);
```

This restricts the variable *TONIC* to 0, 1, 2, 3,

The solution method used is branch-and-bound. It is an intelligent enumeration process that will find a sequence of better and better solutions. As each one is found, the Status Window will be updated with the objective value and a bound on how good a solution might still remain. After the enumeration is complete, various commands from the Reports menu can be used to reveal information about the best solution found.

Let's look at a slightly modified version of the original Enginola problem and see how the GIN specification might help:

```
MAX = 20 * A + 30 * C;
A < 60;
C < 50;
A + 2 * C < 115;
```

Notice the capacity of 115 on the labor constraint (Row 4):

Optimal solution found at step: 1		
Objective value: 2025.000		
Variable	Value	Reduced Cost
A	60.00000	0.0000000
C	27.50000	0.0000000
Row	Slack or Surplus	Dual Price
1	2025.000	1.000000
2	0.0000000	5.000000
3	22.50000	0.0000000
4	0.0000000	15.00000

Note that a fractional quantity is recommended for *C*. If fractional quantities are undesirable, declare *A* and *C* as general integer variables:

```
MAX = 20 * A + 30 * C;
A < 60;
C < 50;
A + 2 * C < 115;
@GIN( A );
@GIN( C );
```

Solving results in the following:

Optimal solution found at step:	4
Objective value:	2020.000
Branch count:	1
Variable	Value
A	59.00000
C	28.00000
Row	Slack or Surplus
1	2020.000
2	1.000000
3	22.00000
4	0.0000000
	Dual Price
	1.000000
	0.0000000
	0.0000000
	0.0000000

2.4.1 Warning for Integer Programs

Although the integer programming (IP) capability is very powerful, it requires skill to use effectively. In contrast to linear programs, just because you can formulate a problem as an integer program, does not mean that it can be solved in very little time. It is very easy to prepare a bad formulation for an essentially easy problem. A bad formulation may require intolerable amounts of computer time to solve. Therefore, you should have access to someone who is experienced in IP formulations if you plan to make use of the IP capability. Good formulations of integer programs are discussed further in Chapter 11, *Formulating and Solving Integer Programs*.

2.5 Solving an Optimization Model

Solving a linear or integer program is a numerically intensive process. We do not discuss the implementation details of the solution algorithms. Writing an efficient solver requires several person-years of effort. For a good introduction to some of the algorithms, see Martin (1999) or Greenberg (1978).

Even though commercial optimization is quite robust, good practice is to avoid using extremely small or extremely large numbers in a formulation. You should try to “scale” the model, so there are no extremely small or large numbers. You should not measure weight in ounces one place and volume in cubic miles somewhere else in the same problem). A rule of thumb is there should be no nonzero coefficient whose absolute value is greater than 100,000 or less than 0.0001. If LINGO feels the model is poorly scaled, it will display a warning. You can usually disregard this warning. However, it is good practice to choose your units of measure appropriately, so this message does not appear.

2.6 Problems

1. Recall the Enginola/Astro/Cosmo problem of the previous chapter. Suppose we add the restriction that only an even number (0, 2, 4...) of Cosmos are allowed. Show how to exploit the `@GIN` command to represent this feature. Note, this kind of restriction sometimes arises in the manufacture of plastic wrap. The product starts out as a long hollow tube. It is flattened and then two resulting edges are cut off to leave you with two flat pieces. Thus, the number of units produced is always a multiple of 2.
2. Using your favorite text editor, enter the Enginola formulation. Save it as a simple, unformatted text file. Start up LINGO, read the model into LINGO, and solve it.
3. Continuing from (2), use LINGO to prepare an output file containing both the formulation and solution. Read this file into your favorite text editor and print it.

3

Analyzing Solutions

3.1 Economic Analysis of Solution Reports

A substantial amount of interesting economic information can be gleaned from the solution report of a model. In addition, optional reports, such as range analysis, can provide further information. The usual use of this information is to do a quick “what if” analysis. The typical kinds of what if questions are:

- (a) What would be the effect of changing a capacity or demand?
- (b) What if a new opportunity becomes available? Is it a worthwhile opportunity?

3.2 Economic Relationship Between Dual Prices and Reduced Costs

The reader hungering for unity in systems may convince himself or herself that a reduced cost is really a dual price born under the wrong sign. Under our convention, the reduced cost of a variable x is really the dual price with the sign reversed on the constraint $x \geq 0$. Recall the reduced cost of the variable x measures the rate at which the solution value deteriorates as x is increased from zero. The dual price on $x \geq 0$ measures the rate at which the solution value improves as the right-hand side (and thus x) is increased from zero.

Our knowledge about reduced costs and dual prices can be restated as:

Reduced cost of an (unused) activity: amount by which profits will decrease if one unit of this activity is forced into the solution.

Dual price of a constraint: one unit reduces amount by which profits will decrease if the availability of the resource associated with this constraint.

We shall argue and illustrate that the reduced cost of an activity is really its net opportunity cost if we “cost out” the activity using the dual prices as charges for resource usage. This sounds like good economic sense. If one unit of an activity is forced into the solution, it effectively reduces the availability of the resources it uses. These resources have an imputed value by way of the dual prices. Therefore, the activity should be charged for the value used. Let’s look at an example and check if the argument works.

3.2.1 The Costing Out Operation: An Illustration

Suppose Enginola is considering adding a video recorder to its product line. Market Research and Engineering estimate the direct profit contribution of a video recorder as \$47 per unit. It would be manufactured on the Astro line and would require 3 hours of labor. If it is produced, it will force the reduction of both Astro production (because it competes for a production line) and Cosmo production (because it competes for labor). Is this tradeoff worthwhile? It looks promising. The video recorder makes more dollars per hour of labor than a Cosmo and it makes more efficient use of Astro capacity than Astros. Recall the dual prices on the Astro and labor capacities in the original solution were \$5 and \$15. If we add this variable to the model, it would have a +47 in the objective function, a +1 in row 2 (the Astro capacity constraint), and a +3 in row 4 (the labor capacity constraint). We can “cost out” an activity or decision variable by charging it for the use of scarce resources. What prices should be charged? The obvious prices to use are the dual prices. The +47 profit contribution can be thought of as a negative cost. The costing out calculations can be arrayed as in the little table below:

Row	Coefficient	Dual Price	Charge
1	-47	1	-47
2	1	+5	+5
3	0	0	0
4	3	15	+45
Total opportunity cost =			+3

Thus, a video recorder has an opportunity cost of \$3. A negative one (-1) is applied to the 47 profit contribution because a profit contribution is effectively a negative cost. The video recorder’s net cost is positive, so it is apparently not worth producing.

The analysis could be stopped at this point, but out of curiosity we’ll formulate the relevant LP and solve it. If V = number of video recorders to produce, then we wish to solve:

$$\begin{aligned} \text{MAX } &= 20 * A + 30 * C + 47 * V; \\ &A + V \leq 60; \\ &C \leq 50; \\ &A + 2 * C + 3 * V \leq 120; \end{aligned}$$

The solution is:

Optimal solution found at step:	1
Objective value:	2100.000
Variable	Value
A	60.000000
C	30.000000
V	0.000000
Reduced Cost	
A	0.000000
C	0.000000
V	3.000000
Row	Slack or Surplus
1	2100.000000
2	0.000000
3	20.000000
4	0.000000
Dual Price	
1	1.000000
2	5.000000
3	0.000000
4	15.000000

Video recorders are not produced. Notice the reduced cost of V is \$3, the value we computed when we “costed out” V . This is an illustration of the following relationship:

The reduced cost of an activity equals the weighted sum of its resource usage rates minus its profit contribution rate, where the weights applied are the dual prices. A

“min” objective is treated as having a dual price of +1. A “max” objective is treated as having a dual price of -1 in the costing out process.

Notice that the dual prices of an LP fully allocate the total profit to all the scarce resources, i.e., for the above example, $5 * 60 + 0 * 50 + 15 * 120 = 2100$.

3.2.2 Dual Prices, LaGrange Multipliers, KKT Conditions, and Activity Costing

When you solve a continuous optimization problem with LINGO or What'sBest!, you can optionally have dual prices reported for each constraint. For simplicity, assume that our objective is to maximize and all constraints are less-than-or-equal-to when all variable expressions are brought to the left-hand side. The dual price of a constraint is then the rate of change of the optimal objective value with respect to the right-hand side of the constraint. This is a generalization to inequality constraints of the idea of a LaGrange multiplier for equality constraints. This idea has been around for more than 100 years. To illustrate, consider the following slightly different, nonlinear problem:

```
[ROW1] MAX = 40*( X+1)^.5 + 30*( Y+1)^.5 + 25*( Z+1)^.5;
[ROW2]           X               + 15* Z   <= 45;
[ROW3]           Y               + Z     <= 45;
[ROW4]           X* X       + 3* Y*Y   + 9 * Z*Z <= 3500;
```

We implicitly assume that $X, Y, Z \geq 0$.

When solved, you get the solution:

Objective value = 440.7100		
Variable	Value	Reduced Cost
X	45.00000	0.0000000
Y	22.17356	0.0000000
Z	0.0000000	0.1140319
Row	Slack or Surplus	Dual Price
ROW2	0.0000000	0.8409353
ROW3	22.82644	0.0000000
ROW4	0.0000000	0.02342115

For example, the dual price of .8409353 on ROW2 implies that if the RHS of ROW2 is increased by a small amount, epsilon, the optimal objective value will increase by about $.8409353 * \text{epsilon}$.

When trying to understand why a particular variable or activity is unused (i.e., at zero), a useful perspective is that of “costing out the activity”. We give the variable “credit” for its incremental contribution to the objective and charge it for its incremental usage of each constraint, where the charging rate applied is the dual price of the constraint. The incremental contribution, or usage, is simply the partial derivative of the LHS with respect to the variable. The costing out of variable Z is illustrated below:

Row	Partial w.r.t Z	Dual price	Total charge
ROW1	12.5	-1	-12.5
ROW2	15	.8409353	12.614029
ROW3	1	0	0
ROW4	0	.02342115	0
Net (Reduced Cost): .11403			

On the other hand, if we do the same costing out for X , we get:

Row	Partial w.r.t X	Dual price	Total charge
ROW1	2.9488391	-1	-2.9488391
ROW2	1	.8409353	.8409353
ROW3	0	0	0
ROW4	90	.02342115	2.107899
Net (Reduced Cost) : 0			

These two computations are illustrations of the Karush/Kuhn/Tucker (KKT) conditions, namely, in an optimal solution:

- a) a variable that has a positive reduced cost will have a value of zero;
- b) a variable that is used (i.e., is strictly positive) will have a reduced cost of zero;
- c) a " \leq " constraint that has a positive dual price will have a slack of zero;
- d) a " \leq " constraint that has strictly positive slack, will have a dual price of zero.

These conditions are sometimes also called complementary slackness conditions.

3.3 Range of Validity of Reduced Costs and Dual Prices

In describing reduced costs and dual prices, we have been careful to limit the changes to “small” changes. For example, if the dual price of a constraint is \$3/hour, then increasing the number of hours available will improve profits by \$3 for each of the first few hours (possibly less than one) added. However, this improvement rate will generally not hold forever. We might expect that, as we make more hours of capacity available, the value (i.e., the dual price) of these hours would not increase and might decrease. This might not be true for all situations, but for LP’s it is true that increasing the right-hand side of a constraint cannot cause the constraint’s dual price to increase. The dual price can only stay the same or decrease.

As we change the right-hand side of an LP, the optimal values of the decision variables may change. However, the dual prices and reduced costs will not change as long as the “character” of the optimal solution does not change. We will say the character changes (mathematicians say the basis changes) when either the set of nonzero variables or the set of binding constraints (i.e., have zero slack) changes. In summary, as we alter the right-hand side, the same dual prices apply as long as the “character” or “basis” does not change.

Most LP programs will optionally supplement the solution report with a range (i.e., sensitivity analysis) report. This report indicates the amounts by which individual right-hand side or objective function coefficients can be changed unilaterally without affecting the character or “basis” of the optimal solution. Recall the previous model:

```

MAX = 20 * A + 30 * C + 47 * V;
      A           + V <= 60;
            C           <= 50;
      A + 2 * C + 3 * V <= 120;
  
```

To obtain the sensitivity report, while in the window with the program, choose *Range* from the *LINGO* menu. The sensitivity report for this problem appears below:

Ranges in which the basis is unchanged:			
	Objective Coefficient Ranges		
Variable	Current Coefficient	Allowable Increase	Allowable Decrease
A	20.00000	INFINITY	3.000000
C	30.00000	10.00000	3.000000
V	47.00000	3.000000	INFINITY

Right-hand Side Ranges			
Row	Current	Allowable	Allowable
	RHS	Increase	Decrease
2	60.00000	60.00000	40.00000
3	50.00000	INFINITY	20.00000
4	120.0000	40.00000	60.00000

Again, we find two sections, one for variables and the second for rows or constraints. The 3 in the *A* row of the report means the profit contribution of *A* could be decreased by up to \$3/unit without affecting the optimal amount of *A* and *C* to produce. This is plausible because one Astro and one Cosmo together make \$50 of profit contribution. If the profit contribution of this pair is decreased by \$3 (to \$47), then a *V* would be just as profitable. Note that one *V* uses the same amount of scarce resources as one Astro and one Cosmo together. The *INFINITY* in the same section of the report means increasing the profitability of *A* by any positive amount would have no effect on the optimal amount of *A* and *C* to produce. This is intuitive because we are already producing *A*'s to their upper limit.

The “allowable decrease” of 3 for variable *C* follows from the same argument as above. The allowable increase of 10 in the *C* row means the profitability of *C* would have to be increased by at least \$10/unit (thus to \$40/unit) before we would consider changing the values of *A* and *C*. Notice at \$40/unit for *C*'s, the profit per hour of labor is the same for both *A* and *C*.

In general, if the *objective function coefficient* of a single variable is *changed within the range* specified in the first section of the range report, then the optimal values of the *decision variables*, *A*, *C*, and *V*, in this case, *will not change*. The dual prices, reduced cost and profitability of the solution, however, may change.

In a complementary sense, if the *right-hand side* of a single constraint is *changed within the range* specified in the second section of the range report, then the *optimal values* of the dual prices and reduced costs *will not change*. However, the values of the decision variables and the profitability of the solution may change.

For example, the second section tells us that, if the right-hand side of row 3 (the constraint $C \leq 50$) is decreased by more than 20, then the dual prices and reduced costs will change. The constraint will then be $C \leq 30$ and the character of the solution changes in that the labor constraint will no longer be binding. The right-hand side of this constraint ($C \leq 50$) could be increased an infinite amount, according to the range report, without affecting the optimal dual prices and reduced costs. This makes sense because there already is excess capacity on the Cosmo line, so adding more capacity should have no effect.

36 Chapter 3 Analyzing Solutions

Let us illustrate some of these concepts by re-solving our three-variable problem with the amount of labor reduced by 61 hours down to 59 hours. The formulation is:

$$\begin{aligned} \text{MAX} &= 20 * A + 30 * C + 47 * V; \\ A &\quad + V \leq 60; \\ C &\quad \leq 50; \\ A + 2 * C + 3 * V &\leq 59; \end{aligned}$$

The solution is:

Optimal solution found at step: 1		
Objective value:		1180.000
Variable	Value	Reduced Cost
A	59.00000	0.0000000
C	0.0000000	10.00000
V	0.0000000	13.00000
Row	Slack or Surplus	Dual Price
1	1180.000	1.000000
2	1.000000	0.0000000
3	50.00000	0.0000000
4	0.0000000	20.00000

Ranges in which the basis is unchanged:

Objective Coefficient Ranges			
Variable	Current Coefficient	Allowable Increase	Allowable Decrease
A	20.00000	INFINITY	4.333333
C	30.00000	10.00000	INFINITY
V	47.00000	13.00000	INFINITY

Right-hand Side Ranges			
Row	Current RHS	Allowable Increase	Allowable Decrease
2	60.00000	INFINITY	1.000000
3	50.00000	INFINITY	50.00000
4	59.00000	1.000000	59.00000

First, note that, with the reduced labor supply, we no longer produce any Cosmos. Their reduced cost is now \$10/unit, which means, if their profitability were increased by \$10 to \$40/unit, then we would start considering their production again. At \$40/unit for Cosmos, both products make equally efficient use of labor.

Also note, since the right-hand side of the labor constraint has reduced by more than 60, most of the dual prices and reduced costs have changed. In particular, the dual price or marginal value of labor is now \$20 per hour. This is because an additional hour of labor would be used to produce one more \$20 Astro. You should be able to convince yourself the marginal value of labor behaves as follows:

Labor Available	Dual Price	Reason
0 to 60 hours	\$20/hour	Each additional hour will be used to produce one \$20 Astro.
60 to 160 hours	\$15/hour	Each additional hour will be used to produce half a \$30 Cosmo.
160 to 280 hours	\$13.5/hour	Give up half an Astro and add half of a <i>V</i> for profit of 0.5 ($-20 + 47$).
More than 280 hours	\$0	No use for additional labor.

In general, the dual price on any constraint will behave in the above stepwise decreasing fashion.

Figures 3.1 and 3.2 give a global view of how total profit is affected by changing either a single objective coefficient or a single right-hand side. The artists in the audience may wish to note that, for a maximization problem:

- a) Optimal total profit as a function of a single objective coefficient always has a bowl shape. Mathematicians say it is a convex function.
- b) Optimal total profit as a function of a single right-hand side value always has an inverted bowl shape. Mathematicians say it is a concave function.

For some problems, as in Figures 3.1 and 3.2, we only see half of the bowl. For minimization problems, the orientation of the bowl in (a) and (b) is simply reversed.

When we solve a problem for a particular objective coefficient or right-hand side value, we obtain a single point on one of these curves. A range report gives us the endpoints of the line segment on which this one point lies.

Figure 3.1 Total Profit vs. Profit Contribution per Unit of Activity V

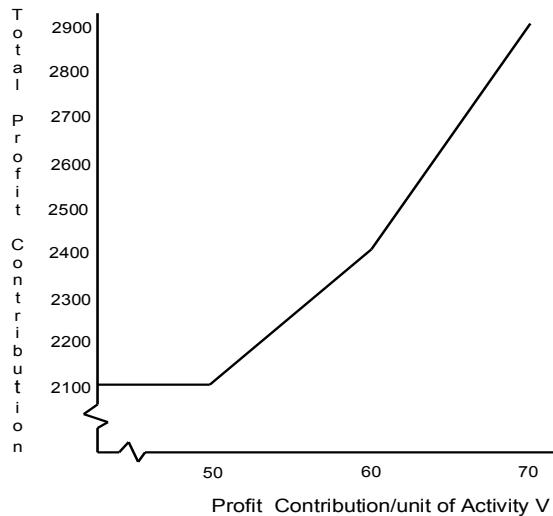
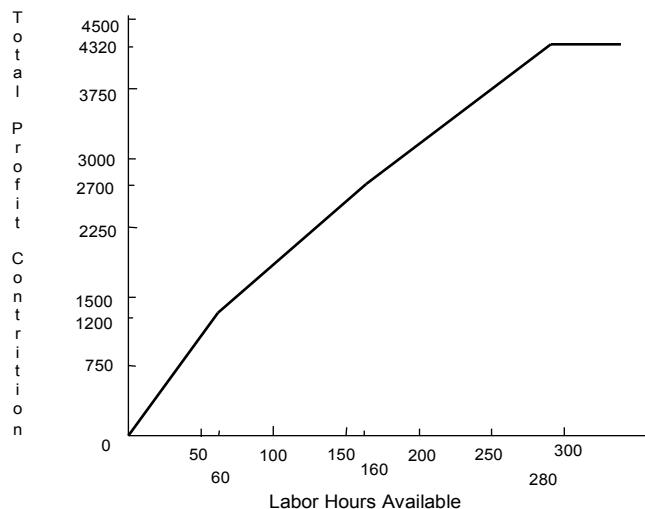


Figure 3.2 Profit vs. Labor Available



3.3.1 Predicting the Effect of Simultaneous Changes in Parameters—The 100% Rule

The information in the range analysis report tells us the effect of changing a single cost or resource parameter. The range report for the Enginola problem is presented as an example:

Ranges in which the basis is unchanged:					
		Objective Coefficient Ranges			
Variable	Current Coefficient	Allowable Increase	Allowable Decrease		
A	20.00000	INFINITY	5.000000		
C	30.00000	10.00000	30.00000		
		Right-hand Side Ranges			
Row	Current RHS	Allowable Increase	Allowable Decrease		
2	60.00000	60.00000	40.00000		
3	50.00000	INFINITY	20.00000		
4	120.0000	40.00000	60.00000		

The report indicates the profit contribution of an Astro could be decreased by as much as \$5/unit without changing the basis. In this case, this means that the optimal solution would still recommend producing 60 Astros and 30 Cosmos.

Suppose, in order to meet competition, we are considering lowering the price of an Astro by \$3/unit and the price of a Cosmo by \$10/unit. Will it still be profitable to produce the same mix? Individually, each of these changes would not change the solution because $3 < 5$ and $10 < 30$.

However, it is not clear these two changes can be made simultaneously. What does your intuition suggest as a rule describing the simultaneous changes that do not change the basis (mix)?

The 100% Rule. You can think of the allowable ranges as slack, which may be used up in changing parameters. It is a fact that any combination of changes will not change the basis if the sum of percentages of slack used is less than 100%. For the simultaneous changes we are contemplating, we have:

$$\left(\frac{3}{5}\right) \times 100 + \left(\frac{10}{30}\right) \times 100 = 60\% + 33\% = 93.3\% < 100\%$$

This satisfies the condition, so the changes can be made without changing the basis. Bradley, Hax, and Magnanti (1977) have dubbed this rule the 100% rule. Since the value of A and C do not change, we can calculate the effect on profits of these changes as $-3 \times 60 - 10 \times 30 = -480$. So, the new profit will be $2100 - 480 = 1620$.

The altered formulation and its solution are:

```
MAX = 17 * A + 20 * C;
      A           <= 60;
      C           <= 50;
      A + 2 * C <= 120;
```

Optimal solution found at step:	1
Objective value:	1620.000
Variable	Value
A	60.00000
C	30.00000
Row	Slack or Surplus
1	1620.000
2	0.0000000
3	20.00000
4	0.0000000
	Dual Price
	1.000000
	7.000000
	0.0000000
	10.00000

3.4 Sensitivity Analysis of the Constraint Coefficients

Sensitivity analysis of the right-hand side and objective function coefficients is somewhat easy to understand because the objective function value changes linearly with modest changes in these coefficients. Unfortunately, the objective function value may change nonlinearly with changes in constraint coefficients. However, there is a very simple formula for approximating the effect of small changes in constraint coefficients. Suppose we wish to examine the effect of decreasing by a small amount e the coefficient of variable j in row i of the LP. The formula is:

$$(\text{improvement in objective value}) \approx (\text{value of variable } j) \times (\text{dual price of row } i) \times e$$

Example: Consider the problem:

```
MAX = (20 * A) + (30 * C);
A <= 65;
C <= 50;
A + 2 * C <= 115;
```

with solution:

Optimal solution found at step:	1
Objective value:	2050.000
Variable	Value
A	65.00000
C	25.00000
Row	Slack or Surplus
1	2050.000
2	0.0000000
3	25.00000
4	0.0000000
	Dual Price
	1.000000
	5.000000
	0.0000000
	15.00000

Now, suppose it is discovered that the coefficient of C in row 4 should have been 2.01, rather than 2. The formula implies the objective value should be decreased by approximately $25 \times 15 \times .01 = 3.75$.

The actual objective value, when this altered problem is solved, is 2046.269, so the actual decrease in objective value is 3.731.

The formula for the effect of a small change in a constraint coefficient makes sense. If the change in the coefficient is small, then the values of all the variables and dual prices should remain essentially unchanged. So, the net effect of changing the 2 to a 2.01 in our problem is effectively to try to use 25 \times .01 additional hours of labor. So, there is effectively 25 \times .01 fewer hours available. However, we have seen that labor is worth \$15 per hour, so the change in profits should be about 25 \times .01 \times 15, which is in agreement with the original formula.

This type of sensitivity analysis gives some guidance in identifying which coefficient should be accurately estimated. If the product of variable j 's value and row i 's dual price is relatively large, then the coefficient in row i for variable j should be accurately estimated if an accurate estimate of total profit is desired.

3.5 The Dual LP Problem, or the Landlord and the Renter

As you formulate models for various problems, you will probably discover that there are several rather different-looking formulations for the same problem. Each formulation may be correct and may be based on taking a different perspective on the problem. An interesting mathematical fact is, for LP problems, there are always two formulations (more accurately, a multiple of two) to a problem. One formulation is arbitrarily called the primal and the other is referred to as the dual. The two different formulations arise from two different perspectives one can take towards a problem. One can think of these two perspectives as the landlord's and the renter's perspectives.

In order to motivate things, consider the following situations. Some textile "manufacturers" in Italy own no manufacturing facilities, but simply rent time as needed from firms that own the appropriate equipment. In the U.S., a similar situation exists in the recycling of some products. Firms that recycle old telephone cable may simply rent time on the stripping machines that are needed to separate the copper from the insulation. This rental process is sometimes called "tolling". In the perfume industry, many of the owners of well-known brands of perfume own no manufacturing facilities, but simply rent time from certain chemical formulation companies to have the perfumes produced as needed. The basic feature of this form of industrial organization is that the owner of the manufacturing resources never owns either the raw materials or the finished product.

Now, suppose you want to produce a product that can use the manufacturing resources of the famous Enginola Company, manufacturer of Astros, Cosmos, and Video Recorders. You would thus like to rent production capacity from Enginola. You need to deduce initial reasonable hourly rates to offer to Enginola for each of its three resources: Astro line capacity, Cosmo line capacity, and labor. These three hourly rates are your decision variables. You in fact would like to rent all the capacity on each of the three resources. Thus, you want to minimize the total charge from renting the entire capacities (60, 50, and 120). If your offer is to succeed, you know your hourly rates must be sufficiently high, so none of Enginola's products are worth producing (e.g., the rental fees foregone by producing an Astro should be greater than 20). These "it's better to rent" conditions constitute the constraints.

Formulating a model for this problem, we define the variables as follows:

- PA = price per unit to be offered for Astro line capacity,
- PC = price per unit to be offered for Cosmo line capacity,
- PL = price per unit to be offered for labor capacity.

Then, the appropriate model is:

The Dual Problem:

```
MIN = 60 * PA + 50 * PC + 120 * PL;
!ASTRO; PA + PL > 20;
!COSMO; PC + 2*PL > 30;
!VR;     PA + 3 * PL > 47;
```

The three constraints force the prices to be high enough, so it is not profitable for Enginola to produce any of its products.

The solution is:

Optimal solution found at step: 2		
Objective value:		2100.000
Variable	Value	Reduced Cost
PA	5.000000	0.0000000
PC	0.0000000	20.00000
PL	15.00000	0.0000000
Row	Slack or Surplus	Dual Price
1	2100.000	1.000000
2	0.0000000	-60.00000
3	0.0000000	-30.00000
4	3.000000	0.0000000

Recall the original, three-product Enginola problem was:

The Primal Problem:

$$\begin{aligned} \text{MAX} &= 20 * A + 30 * C + 47 * V; \\ A + &V \leq 60; \\ C &\leq 50; \\ A + 2 * C + 3 * V &\leq 120; \end{aligned}$$

with solution:

Optimal solution found at step: 1		
Objective value:		2100.000
Variable	Value	Reduced Cost
A	60.00000	0.0000000
C	30.00000	0.0000000
V	0.0000000	3.000000
Row	Slack or Surplus	Dual Price
1	2100.000	1.000000
2	0.0000000	5.000000
3	20.00000	0.0000000
4	0.0000000	15.00000

Notice the two solutions are essentially the same, except prices and decision variables are reversed. In particular, note the price the renter should pay is exactly the same as Enginola's original profit contribution. This "Minimize the rental cost of the resources, subject to all activities being unprofitable" model is said to be the dual problem of the original "Maximize the total profit, subject to not exceeding any resource availabilities" model. The equivalence between the two solutions shown above always holds. Upon closer scrutiny, you should also notice the dual formulation is essentially the primal formulation "stood on its ear," or its transpose, in fancier terminology.

Why might the dual model be of interest? The computational difficulty of an LP is approximately proportional to m^2n , where m = number of rows and n = number of columns. If the number of rows in the dual is substantially smaller than the number of rows in the primal, then one may prefer to solve the dual.

Additionally, certain constraints, such as simple upper bounds (e.g., $x \leq 1$) are computationally less expensive than arbitrary constraints. If the dual contains only a small number of arbitrary constraints, then it may be easier to solve the dual even though it may have a large number of simple constraints.

The term “dual price” arose because the marginal price information to which this term is applied is a decision variable value in the dual problem.

We can summarize the idea of dual problems as follows. If the original or primal problem has a Maximize objective with \leq constraints, then its dual has a Minimize objective with \geq constraints. The dual has one variable for each constraint in the primal and one constraint for each variable in the primal. The objective coefficient of the k th variable of the dual is the right-hand side of the k th constraint in the primal. The right-hand side of constraint k in the dual is equal to the objective coefficient of variable k in the primal. Similarly, the coefficient in row i of variable j in the dual equals the coefficient in row j of variable i in the primal.

In order to convert all constraints in a problem to the same type, so one can apply the above, note the following two transformations:

- (1) The constraint $2x + 3y = 5$ is equivalent to the constraints $2x + 3y \geq 5$ and $2x + 3y \leq 5$;
- (2) The constraint $2x + 3y \geq 5$ is equivalent to $-2x - 3y \leq -5$.

Example: Write the dual of the following problem:

$$\begin{aligned} & \text{Maximize } 4x - 2y \\ & \text{subject to } 2x + 6y \leq 12 \\ & \quad 3x - 2y = 1 \\ & \quad 4x + 2y \geq 5 \end{aligned}$$

Using transformations (1) and (2) above, we can rewrite this as:

$$\begin{aligned} & \text{Maximize } 4x - 2y \\ & \text{subject to } 2x + 6y \leq 12 \\ & \quad 3x - 2y \leq 1 \\ & \quad -3x + 2y \leq -1 \\ & \quad -4x - 2y \leq -5 \end{aligned}$$

Introducing the dual variables r , s , t , and u , corresponding to the four constraints, we can write the dual as:

$$\begin{aligned} & \text{Minimize } 12r + s - t - 5u \\ & \text{subject to } 2r + 3s - 3t - 4u \geq 4 \\ & \quad 6r - 2s + 2t - 2u \geq -2 \end{aligned}$$

3.6 Problems

1. The Enginola Company is considering introducing a new TV set, the Quasi. The expected profit contribution is \$25 per unit. This unit is produced on the Astro line. Production of one Quasi requires 1.6 hours of labor. Using only the original solution below, determine whether it is worthwhile to produce any Quasi's, assuming no change in labor and Astro line capacity.

The original Enginola problem with solution is below.

```
MAX = 20 * A + 30 * C;
A      <= 60;
C      <= 50;
A + 2 * C <= 120;
```

```

Optimal solution found at step:      1
Objective value:                  2100.000
Variable          Value       Reduced Cost
  A            60.00000    0.0000000
  C            30.00000    0.0000000
Row    Slack or Surplus     Dual Price
  1        2100.000       1.000000
  2        0.0000000      5.000000
  3        20.00000      0.0000000
  4        0.0000000      15.00000

```

2. The Judson Corporation has acquired 100 lots on which it is about to build homes. Two styles of homes are to be built, the “*Cape Cod*” and the “*Ranch Home*”. Judson wishes to build these 100 homes over the next nine months. During this time, Judson will have available 13,000 man-hours of bricklayer labor and 12,000 hours of carpenter labor. Each *Cape Cod* requires 200 man-hours of carpentry labor and 50 man-hours of bricklayer labor. Each *Ranch Home* requires 120 hours of bricklayer labor and 100 man-hours of carpentry. The profit contribution of a *Cape Cod* is projected to be \$5,100, whereas that of a *Ranch Home* is projected at \$5,000. When formulated as an LP and solved, the problem is as follows:

```

MAX = 5100 * C + 5000 * R;
C +
R < 100;
200 * C + 100 * R < 12000;
50 * C + 120 * R < 13000;

```

```

Optimal solution found at step:      0
Objective value:                  502000.0
Variable          Value       Reduced Cost
  C            20.00000    0.0000000
  R            80.00000    0.0000000
Row    Slack or Surplus     Dual Price
  1        502000.0       1.000000
  2        0.0000000      4900.000
  3        0.0000000      1.000000
  4        2400.000      0.0000000

```

Ranges in which the basis is unchanged:

Objective Coefficient Ranges			
	Current	Allowable Increase	Allowable Decrease
Variable	Coefficient		
C	5100.000	4900.000	100.0000
R	5000.000	100.0000	2450.000

Right-hand Side Ranges			
	Current	Allowable Increase	Allowable Decrease
Row	RHS		
2	100.0000	12.63158	40.00000
3	12000.00	8000.000	2000.000
4	13000.00	INFINITY	2400.000

- (a) A gentleman who owns 15 vacant lots adjacent to Judson’s 100 lots needs some money quickly and offers to sell his 15 lots for \$60,000. Should Judson buy? What assumptions are you making?

- (b) One of Judson's salesmen who is a native of Massachusetts feels certain he could sell the *Cape Cods* for \$2,000 more each than Judson is currently projecting. Should Judson change its planned mix of homes? What assumptions are inherent in your recommendation?
3. Jack Mazzola is an industrial engineer with the Enginola Company. He has discovered a way of reducing the amount of labor used in the manufacture of a Cosmo TV set from 2 hours per set to 1.92 hours per set by replacing one of the assembled portions of the set with an integrated circuit chip. It is not clear at the moment what this chip will cost. Based solely on the solution report below (i.e., do not solve another LP), answer the following questions:

- (a) Assuming labor supply is fixed, what is the approximate value of one of these chips in the short run?
- (b) Give an estimate of the approximate increase in profit contribution per day of this change, exclusive of chip cost.

```
MAX = 20 * A + 30 * C;
      A      <= 60;
      C      <= 50;
      A + 2 * C <= 120;
```

Optimal solution found at step:	1		
Objective value:	2100.000		
Variable	Value	Reduced Cost	
A	60.00000	0.0000000	
C	30.00000	0.0000000	
Row	Slack or Surplus	Dual Price	
1	2100.000	1.000000	
2	0.0000000	5.000000	
3	20.00000	0.0000000	
4	0.0000000	15.00000	
Right-hand Side Ranges			
Row	Current RHS	Allowable Increase	Allowable Decrease
2	60.00000	60.00000	40.00000
3	50.00000	INFINITY	20.00000
4	120.0000	40.00000	60.00000

4. The Bug product has a profit contribution of \$4100 per unit and requires 4 hours in the foundry department and 2 hours in the assembly department. The SuperBug has a profit contribution of \$5900 per unit and requires 2 hours in the foundry and 3 hours in assembly. The availabilities in foundry and assembly are 160 hours and 180 hours, respectively. Each hour used in each of foundry and assembly costs \$90 and \$60, respectively. The following is an LP formulation for maximizing profit contribution in this situation:

```
MAX = 4100 * B + 5900 * S - 90 * F - 60 * A;
      4 * B +      2 * S -      F      = 0;
      2 * B +      3 * S      - A      = 0;
                           F      <= 160;
                           A      <= 180;
```

Following is an optimal solution report printed on a typewriter that skipped some sections of the report.

Objective value:

Variable	Value	Reduced Cost
B		73.33325
S	60.00000	
F	120.0000	0.0000000
A	180.0000	0.0000000
Row	Slack or Surplus	Dual Price
1	332400.0	1.000000
2	0.0000000	
3		1906.667
4		0.0000000
5	0.0000000	1846.667

Fill in the missing parts, using just the available information (i.e., without re-solving the model on the computer).

5. Suppose the capacities in the Enginola problem were: Astro line capacity = 45; labor capacity = 100.
 - (a) Allow the labor capacity to vary from 0 to 200 and plot:
 - Dual price of labor as a function of labor capacity.
 - Total profit as a function of labor capacity.
 - (b) Allow the profit contribution/unit of Astros to vary from 0 to 50 and plot:
 - Number of Astros to produce as a function of profit/unit.
 - Total profit as a function of profit/unit.
6. Write the dual problem of the following problem:

$$\begin{aligned} \text{Minimize } & 12q + 5r + 3s \\ \text{subject to } & q + 2r + 4s \geq 6 \\ & 5q + 6r - 7s \leq 5 \\ & 8q - 9r + 11s = 10 \end{aligned}$$

7. The energetic folks at Enginola, Inc. have not been idle. The R & D department has given some more attention to the proposed digital recorder product (code name R) and enhanced it so much that everyone agrees it could be sold for a profit contribution of \$79 per unit. Unfortunately, its production still requires one unit of capacity on both the A(stro) and C(osmo) lines. Even worse, it now requires four hours of labor. The Marketing folks have spread the good word about the Astro and Cosmo products, so a price increase has been made possible. Industrial Engineering has been able to increase the capacity of the two lines. The new ex-marine heading Human Resources has been able to hire a few more good people, so the labor capacity has increased to 135 hours. The net result is that the relevant model is now:

```

MAX = 23 * A + 38 * C + 79 * R;
      A           + R <= 75;
      C           + R <= 65;
      A + 2 * C + 4 * R <= 135;
END
  
```

Without resorting to a computer, answer the following questions, supporting each answer with a one- or two-sentence economic argument that might be understood by your spouse or “significant other.”

- (a) How many *A*'s should be produced?
- (b) How many *C*'s should be produced?
- (c) How many *R*'s should be produced?
- (d) What is the marginal value of an additional hour of labor?
- (e) What is the marginal value/unit of additional capacity on the *A* line?
- (f) What is the marginal value per unit of additional capacity on the *C* line?

4

The Model Formulation Process

Count what is countable, measure what is measurable, and what is not measurable, make measurable.

Galileo Galilei(1564-1642)

4.1 The Overall Process

In using any kind of analytical or modeling approach for attacking a problem, there are five major steps:

- 1) Understanding the real problem.
- 2) Formulating a model of the problem.
- 3) Gathering and generating the input data for the model (e.g., per unit costs to be used, etc.).
- 4) Solving or running the model.
- 5) Implementing and interpreting the solution in the real world.

In general, there is a certain amount of iteration over the five (e.g., one does not develop the most appropriate model the first time around). Of the above, the easiest is the solving of the model on the computer. This is not because it is intrinsically easiest, but because it is the most susceptible to mathematical analysis. Steps 1, 3, and 5 are, if not the most difficult, at least the most time consuming. Success with these steps depends to a large extent upon being very familiar with the organization involved (e.g., knowing who knows what the real production rate is on the punch press machine). Step 2 requires the most analytical skill. Steps 1 and 5 require the most people skills.

Formulating good models is an art bordering on a science. The artistic ability is in developing simple models that are nevertheless good approximations of reality. We shall see that there are a number of classes of problems that are well approximated by optimization models.

With all of the above comments in mind, we will devote most of the discussion to formulation of optimization models, stating what universal truths seem to apply for steps (3) and (5), and giving an introduction to the mechanics of step (4).

4.2 Approaches to Model Formulation

We take two approaches to formulating models:

- 1) Template approach,
- 2) Constructive approach.

The constructive approach is the more fundamental and general. However, readers with less analytic skill may prefer the template approach. The latter is essentially a “model in a can” approach. In this approach, examples of standard applications are illustrated in substantial detail. If you have a problem that closely resembles one of these “template” models, you may be able to adjust it to your situation by making modest changes to the template model. The advantage of this approach is that the user may not need much technical background if there is a template model that closely fits the real situation.

4.3 The Template Approach

You may feel more comfortable and confident in your ability to structure problems if you have a classification of “template” problems to which you can relate new problems you encounter. We will present a classification of about a half dozen different categories of problems. In practice, a large real problem you encounter will not fit a single template model exactly, but might require a combination of two or more of the categories. The classification is not exhaustive, so you may encounter or develop models that seem to fit none of these templates.

4.3.1 Product Mix Problems

Product mix problems are the problem types typically encountered in introductory LP texts. There are a collection of products that can be sold and a finite set of resources from which these products are made. Associated with each product are a profit contribution rate and a set of resource usage rates. The objective is to find a mix of products (amount of each product) that maximizes profit, subject to not using more resources than are available.

These problems are always of the form “Maximize profit subject to less-than-or-equal-to constraints”.

4.3.2 Covering, Staffing, and Cutting Stock Problems

Covering, staffing, and cutting stock problems are complementary (in the jargon, they are called dual) to product mix problems in that their form is “Minimize cost subject to greater-than-or-equal-to constraints”. The variables in this case might correspond to the number of people hired for various shifts during the day. The constraints arise from the fact that the mix of variables chosen must “cover” the requirements during each hour of the day.

4.3.3 Blending Problems

Blending problems arise in the food, feed, metals, and oil refining industries. The problem is to mix or blend a collection of raw materials (e.g., different types of meats, cereal grains, or crude oils) into a finished product (e.g., sausage, dog food, or gasoline). The cost per unit of the finished product is minimized and it is subject to satisfying certain quality constraints (e.g., percent protein \geq 15 percent).

4.3.4 Multiperiod Planning Problems

Multiperiod planning problems constitute perhaps the most important class of models. These models take into account the fact that the decisions made in this period partially determine which decisions are allowable in future periods. The submodel used each period may be a product mix problem, a blending problem, or some other type. These submodels are usually tied together by means of inventory variables (e.g., the inventory of raw materials, finished goods, cash, or loans outstanding) that are carried from one period to the next.

4.3.5 Network, Distribution, and PERT/CPM Models

Network LP models warrant special attention for two reasons: (a) they have a particularly simple form, which makes them easy to describe as a graph or network, and (b) specialized and efficient solution procedures exist for solving them. They, therefore, tend to be easier to explain and comprehend. Network LPs frequently arise from problems of product distribution. Any enterprise producing a product at several locations and distributing it to many customers may find a network LP relevant. Large problems of this type may be solved rapidly by the specialized procedures.

One of the simplest network problems is finding the shortest route from one point in a network to another. A slight variation on this problem, finding the longest route, happens to be an important component of the project management tools PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method).

Close cousins of network models are input/output and vertically integrated models. General Motors, for example, makes engines in certain plants. These engines might be sold directly to customers, such as industrial equipment manufacturers, or the engines may be used in GM's own cars and trucks. Such a company is said to be vertically integrated. In a vertically integrated model, there is usually one constraint for each type of intermediate product. The constraint mathematically enforces the basic law of physics that the amount used of an intermediate product by various processes cannot exceed the amount of this product produced by other processes. There is usually one decision variable for each type of process available.

If one expands one's perspective to the entire economy, then the models considered tend to be similar to the input/output model popularized by Wassily Leontief (1951). Each industry is described by the input products required and the output products produced. These outputs may in turn be inputs to other industries. The problem is to determine appropriate levels at which each industry should be operated in order to satisfy specific consumption requirements.

4.3.6 Multiperiod Planning Problems with Random Elements

One of the fundamental assumptions of optimization models is that all input data are known with certainty. There are situations, however, where certain key data are highly random. For example, when an oil company makes its fuel oil production decisions for the coming winter, the demand for that fuel oil is very much a random variable. If, however, the distribution probabilities for all the random variables are known, then there is a modeling technique for converting a problem that is an optimization model, except for the random elements, into an equivalent, although possibly larger, deterministic optimization model. Such models are sometimes called stochastic programs.

4.3.7 Financial Portfolio Models

An important application of optimization in the last ten years has been in the design of financial investment portfolios. In its simplest form, it is concerned with how much to invest in a collection of risky investments, so that a good compromise is struck between a high expected return and a low risk.

More complicated applications of this idea are concerned with investing so as to track some popular financial index, such as the S&P 500.

4.3.8 Game Theory Models

Game theory is concerned with the analysis of competitive situations. In its simplest form, a game consists of two players, each of whom has available to them a set of possible decisions. Each player must choose a strategy for making a decision in ignorance of the other player's choice. Some time after a decision is made, each player receives a payoff that depends on which combination of decisions was made. The problem of determining each player's optimal strategy can be formulated as a linear program.

Not all problems you encounter will fit into one of the above categories. Many problems will be combinations of the above types. For example, in a multiperiod planning problem, the single period subproblems may be product mix or blending problems.

4.4 Constructive Approach to Model Formulation

The constructive approach is a set of guidelines for constructing a model from the ground up. This approach requires somewhat more analytical skill, but the rules apply to any situation you are trying to model. The odds are low you will find a template model that exactly matches your real situation. In practice, a combination of these two approaches is needed.

For the constructive approach, we suggest the following three-step approach for constructing a model, which, with apologies to Sam Savage, might be called the ABC's of modeling:

- A. Identify and define the decision variables or *Adjustable cells*. Defining a decision variable includes specifying the units in which it is measured (e.g., tons, hours, etc.). One way of trying to deduce the decision variables is to ask the question: What should be the format of a report that gives a solution to this problem? (For example, the numbers that constitute an answer are: the amount to produce of each product and the amount to use of each ingredient.) The cells in this report are the decision variables.
- B. Define how we measure *Best*. More officially, define our objective or criterion function, including the units in which it is measured. Among useable or feasible solutions, how would preference/goodness (e.g., profit) be measured?
- C. Specify the *Constraints*, including the units in which each is measured. A way to think about constraints is as follows: Given a purported solution to a problem, what numeric checks would you perform to check the validity of the solution?

The majority of the constraints in most problems can be thought of as *sources-equals-uses* constraints. Another common kind of constraint is the *definitional* or *accounting* constraint. Sometimes the distinction between the two is arbitrary. Consider a production setting where we: *i*) start with some beginning inventory of some commodity, *ii*) produce some of that commodity, *iii*) sell some of the commodity, and *iv*) leave some of the commodity in ending inventory. From the sources-equals-uses perspective, we might write:

$$\text{beginning inventory} + \text{production} = \text{sales} + \text{ending inventory}.$$

From the definitional perspective, if we were thinking of how ending inventory is defined, we would write:

$$\text{ending inventory} = (\text{beginning inventory} + \text{production}) - \text{sales}.$$

The two perspectives are in fact mathematically equivalent.

For any application, it is useful to do each of the above in words first. In order to illustrate these ideas, consider the situation in the following example.

4.4.1 Example

Deglo Toys has been manufacturing a line of precision building blocks for children for a number of years. Deglo is faced with a standard end-of-the-year problem known as the build-out problem. It is about to introduce a new line of glow-in-the-dark building blocks. Thus, they would like to deplete their old-technology inventories before introducing the new line. The old inventories consist of 19,900 4-dimple blocks and 29,700 8-dimple blocks. These inventories can be sold off in the form of two different kits: the Master Builder and the Empire Builder. The objective is to maximize the revenue from the sale of these two kits. The Master kit sells for \$16.95, and the Empire kit sells for \$24.95. The Master kit is composed of 30 4-dimple blocks plus 40 8-dimple blocks. The Empire kit is composed of 40 4-dimple blocks plus 85 8-dimple blocks. What is an appropriate model of this problem?

4.4.2 Formulating Our Example Problem

The process for our example problem would be as follows:

- The essential decision variables are:
 M = number of master builder kits to assemble and
 E = number of empire builder kits to assemble.
- The objective function is to maximize sales revenue (i.e., Maximize $16.95M + 24.95E$).
- If someone gave us a proposed solution (i.e., values for M and E), we would check its feasibility by checking that:
 - the number of 4-dimple blocks used $\leq 19,900$ and
 - the number of 8-dimple blocks used $\leq 29,700$.

Symbolically, or algebraically, this is:

$$30M + 40E \leq 19,900$$

$$40M + 85E \leq 29,700$$

In LINGO form, the formulation is:

```
MAX = 16.95 * M + 24.95 * E;
      30 * M      + 40 * E <= 19900;
      40 * M      + 85 * E <= 29700;
```

with solution:

Optimal solution found at step:	0	
Objective value:	11478.50	
Variable	Value	Reduced Cost
M	530.0000	0.0000000
E	100.0000	0.0000000
Row	Slack or Surplus	Dual Price
1	11478.50	1.000000
2	0.0000000	0.4660526
3	0.0000000	0.7421052E-01

Thus, we should produce 530 Master Builders and 100 Empire Builders.

4.5 Choosing Costs Correctly

Choosing costs and profit contribution coefficients in the objective requires some care. In many firms, cost data may not be available at the detailed level required in an optimization model. If available, the “official” cost coefficients may be inappropriate for the application at hand.

The basic rule is fairly simple: The cost coefficient of a variable should be the rate of change of the total cost as the variable changes. We will discuss the various temptations to violate this rule. The two major temptations are sunk costs and joint costs.

4.5.1 Sunk vs. Variable Costs

A *sunk cost* is a cost that has already been incurred or committed to, although not necessarily paid. A *variable cost* is a cost that varies with some activity level. Sunk costs should not appear in any coefficient of a decision variable. Whether a cost is sunk or variable depends closely upon the length of our planning horizon. A general rule is that: In the short run, all costs are sunk, while all costs are variable in the long run. The following example illustrates.

Sunk and Variable Cost Example

A firm prepared a profit contribution table for two of its products, X and Y :

<u>Product:</u>	<u>X</u>	<u>Y</u>
Selling price/unit	\$1000	\$1000
Material cost/unit	\$200	\$300
Labor cost/unit	<u>\$495</u>	<u>\$300</u>
Net Profit contribution	\$305	\$400

These two products use a common assembly facility that has a daily capacity of 80 units. Product specific production facilities limit the daily production of X to 40 units and Y to 60 units. The hourly wage in the company is \$15/hour for all labor. The obvious model is:

$$\begin{aligned} \text{Max} = & 305 * X + 400 * Y; \\ X & \leq 40; \\ Y & \leq 60; \\ X + Y & \leq 80; \end{aligned}$$

The solution is to produce 60 Y 's and 20 X 's. At \$15 per hour, the total labor required by this solution is $20 \times 495/15 + 60 \times 300/15 = 1860$ hours per day.

Now, let us consider some possible additional details or variations of the above situation. Some firms, such as some automobile manufacturers, have had labor contracts that effectively guarantee a job to a fixed number of employees during the term of the contract (e.g., one year). If the above model is being used to decide how many employees to hire and commit to before signing the contract, then the \$15/hour used above is perhaps appropriate, although it may be too low. In the U.S., the employer also must pay Social Security and Medicare taxes that add close to 8% to the labor bill. In addition, the employer typically also covers the cost of supplemental health insurance for the employee, so the cost of labor is probably closer to \$20 per hour rather than \$15.

Once the contract is signed, however, the labor costs then become sunk, but we now have a constraint that we can use at most 1860 hours of labor per day. The variable profit contributions are now:

<u>Product:</u>	<u>X</u>	<u>Y</u>
Selling price/unit	\$1000	\$1000
Material cost/unit	\$200	\$300
Net Profit contribution	\$800	\$700

Now, X is the more profitable product. Before we jump to the conclusion that we should now produce 40 X 's and 40 Y 's, we must recall that labor capacity is now fixed. The proper, short term, model is now:

```
MAX = 800 * X + 700 * Y;
      X           <= 40;
                  Y <= 60;
      X           + Y <= 80;
33 * X + 20 * Y <= 1860;
```

with solution:

Optimal solution found at step: 1		
Objective value: 58000.00		
Variable	Value	Reduced Cost
X	20.00000	0.0000000
Y	60.00000	0.0000000
Row	Slack or Surplus	Dual Price
1	58000.00	1.000000
2	20.00000	0.0000000
3	0.0000000	215.1515
4	0.0000000	0.0000000
5	0.0000000	24.24242

Therefore, we still produce the same mix of products.

Now, suppose in order to be competitive, the selling price of X must be dropped by \$350 to \$650. Also, we still have our labor contract that says we may use up to and must pay for all of 1860 hours of labor per day. The correct model is:

```
Max = 450 * X + 700 * Y;
      X           <= 40;
                  Y <= 60;
      X           + Y <= 80;
33 * X + 20 * Y <= 1860;
```

with still the same solution of $X = 20$ and $Y = 60$. If we (incorrectly) charge for labor, however, the model is:

```
Max = - 45 * X + 400 * Y;
      X           <= 40;
                  Y <= 60;
      X           + Y <= 80;
33 * X + 20 * Y <= 1860;
```

and we would incorrectly conclude that X should not be produced.

There are many planning situations similar to the above. For example, an airline or a trucking firm may use essentially the same model for long-range fleet sizing decisions as for daily fleet routing decision. When solving the long-term fleet sizing decision, the cost of capital should be included in the daily cost of having a vehicle. On the other hand, when making short-run routing decisions, the cost of capital should not be included in the daily cost of a vehicle. However, the number of vehicles used is constrained to be no greater than the fleet size chosen in the long-term plan. Only operating costs that vary with the amount of usage of the vehicle should be included when solving the short-term model.

4.5.2 Joint Products

We say we have joint products or byproducts if a single process produces several products. The key feature is that, if you run the process in question, you unavoidably get some amount of each of the joint products. Some examples are:

Process	Joint Products
Crude oil distillation	gasoline, oil, kerosene, tar
Raw milk processing	whole milk, skim milk, 2%, cream, yogurt
Meat processing	light meat, dark meat, steak, chuck roast
Semi-conductor manufacturing	high speed chips, low speed chips
Mining of precious metal ore	gold, silver, copper
Sales calls	sales of various products in product line

There is a temptation, perhaps even a requirement by taxing authorities, that the cost of the joint process be fully allocated to the output products. The important point is that, for decision-making purposes, this allocation serves no purpose. It should be avoided. The proper way to model a joint product process is to have a separate decision variable for each output product, and a decision variable for the joint production process. Costs and revenues should be applied to their associated decision variables (e.g., the cost of distillation should be associated with the decision variable of how much crude to distill). The fact that, if you want to produce gasoline, then you must incur the cost of distillation is taken care of by the constraints. Let us illustrate with an example.

Joint Cost Example

The Chartreuse Company (CC) raises pumpkins. It costs \$800 to plant, harvest and sort a ton of raw pumpkins. CC has capacity to plant and harvest 150 tons of pumpkins. In spite of CC's best efforts at genetic engineering, harvested pumpkins fall equally into three classes of pumpkins of increasing quality: Good, Premium, and Exquisite. Once sorted, it costs \$100 per ton to get each of the classes ready for market. Alternatively, pumpkins from any class can be discarded at zero additional cost. Prices have dropped recently, so there is concern about whether it is profitable to sell all grades of pumpkins. Current selling prices per ton for the three grades are: \$700, \$1100, and \$2200. How much should be processed and sold of each grade?

A proper model is:

```

MAX = ( 700 - 100) * G + (1100 - 100) * P + (2200 -
100) * E - 800 * R;
R <= 150;
G <= .3333333 * R;
P <= .3333333 * R;
E <= .3333333 * R;

```

With solution:

Optimal solution found at step: 0		
Objective value:		65000.0
Variable	Value	Reduced Cost
G	50.00000	0.0000000
P	50.00000	0.0000000
E	50.00000	0.0000000
R	150.0000	0.0000000
Row	Slack or Surplus	Dual Price
1	65000.00	1.000000
2	0.0000000	433.3333
3	0.0000000	600.0000
4	0.0000000	1000.000
5	0.0000000	2100.000

There is a temptation to allocate the cost of planting, harvesting and sorting, over all three grades to get the model:

```

MAX = ( 700 - 100 - 2400/3) * G + (1100 - 100 -
2400/3) * P + (2200 - 100 - 2400/3) * E ;
G <= .333333 * 150;
P <= .333333 * 150;
E <= .333333 * 150;

```

Given their (apparent) negative profit contribution in the above model, good pumpkins will not be produced. If we then allocate the planting, harvesting, and sorting costs over just P and E , we get:

```

MAX = (1100 - 100 - 2400/2) * P + (2200 - 100 -
2400/2) * E;
G <= .333333 * 150;
P <= .333333 * 150;
E <= .333333 * 150;

```

Now, of course, Premium grade is not worth producing. This leaves the Exquisite grade to carry the full cost of planting, harvesting, and sorting, and then we see it is not worth producing. Thus, even though we started with a profitable enterprise, blind use of allocation of joint costs caused us to quit the profitable business. The moral to the story is to not do cost allocation.

4.5.3 Book Value vs. Market Value

A common problem in formulating an optimization model for decisionmaking is what cost should be attached to product that is used from inventory. A typical accounting system will carry a book value for product in inventory. The temptation is to use this readily available number as the cost of using product from inventory. For example, suppose you are a gasoline distributor who bought 10,000 gallons of Regular gasoline last month for \$2.77 per gallon. Due to unforeseen events, this month you still have 5,000 gallons of that Regular gasoline in inventory. Now the market price for Regular gasoline has dropped to \$2.70 per gallon, and you are contemplating your production and market operations for this month. How much should you charge yourself for the use of this Regular in inventory? One person might argue that the purchase is now a sunk cost so we should charge ourselves 0. Others might argue that proper “Accounting” says we should charge the book value, \$2.77/gallon. Which is it? The simple quick answer is that for decision making purposes, book value should always be disregarded, except when required by law for the calculation of taxes. Material in

inventory should be treated as having zero cost, however, you should completely enumerate all possible options of what you can do with this inventory, including selling it on the open market.

It helps to clarify issues by completing all the details for our little example and explicitly defining decision variables for all the possible actions available. You can buy or sell Regular in unlimited amounts this month for \$2.70/gallon, however, it costs you \$0.01/gallon in transportation and transaction costs for any gasoline you buy to get it onto your property. Similarly, for any gasoline you sell, there is a transaction cost of \$0.02 per gallon. What can be done with Regular gasoline? It can be sold directly, or it can be blended in equal proportions with Premium gasoline to produce Midgrade gasoline. You have one customer who is willing to pay \$2.82/gallon of Midgrade delivered to his door for up to 6000 gallons, and a second customer who is willing to pay \$2.80/gallon of Midgrade delivered to his door for up to 8000 gallons. Premium gasoline can be purchased in unlimited amounts for \$2.90/gallon. What should we do with our Regular gasoline: nothing, sell it back to the market, buy some Premium to blend with Regular (perhaps even buying more Regular) to sell to customer 1, to customer 2? Following the ABC's of optimization, step A is to define our decision variables: RB = gallons of additional Regular gasoline bought on the market this month, RS = gallons of Regular directly sold on the market this month, PB = gallons of Premium bought, $MS1$ = gallons of Midgrade sold to customer 1, and $MS2$ = gallons of Midgrade sold to customer 2. Step B, the objective function is to maximize revenues minus costs. Step C is to specify the constraints. The two main constraints are the "Sources EQual Uses" constraints for Regular and Premium. A formulation is given below. Recall that a gallon of Midgrade uses a half gallon of Regular and half gallon of Premium. To make the solution report easier to understand, we have given a [row name] to each constraint.

```

!Maximize revenues - costs;
MAX = (2.70 - .02)*RS + (2.82 - .02)*MS1 + (2.80-.02)*MS2
      - (2.70 + .01)*RB - (2.90 + .01)*PB;
!Sources = uses for Regular and Premium;
[SEQUR]      5000 + RB = RS + .5*(MS1 + MS2);
[SEQUP]      PB =      .5*(MS1 + MS2);
!Upper limits on amount we can sell;
[UL1]          MS1 <= 6000;
[UL2]          MS2 <= 8000;

```

Notice there is no explicit charge for Regular in inventory. The book value of \$2.77 appears nowhere in the formulation. Inventory is treated as a sunk cost or free good, however, we have included the option to sell it directly. Thus, using Regular to blend Midgrade must compete with simply selling the Regular directly at the current market price. A solution is:

Objective value: 13430.00		
Variable	Value	Reduced Cost
RS	2000.000	0.000000
MS1	6000.000	0.000000
MS2	0.000	0.015000
RB	0.000	0.030000
PB	3000.000	0.000000

Row	Slack or Surplus	Dual Price
1	13430.000	1.000000
SEQUR	0.000	-2.680000
SEQUP	0.000	-2.910000
UL1	0.000	0.005000

UL2	8000.000	0.000000
-----	----------	----------

Thus, it is more profitable to blend the Regular inventory with Premium to sell it to customer 1 than to sell it directly to the market, however, selling Regular directly back to the market is more profitable than selling to customer 2 blended into Midgrade.

4.6 Common Errors in Formulating Models

When you develop a first formulation of some real problem, the formulation may contain errors or bugs. These errors will fall into the following categories:

- A. Simple typographical errors;
- B. Fundamental errors of formulation;
- C. Errors of approximation.

The first two categories of errors are easy to correct once they are identified. In principle, category *A* errors are easy to identify because they are clerical in nature. In a large model, however, tracking them down may be a difficult search problem. Category *B* errors are more fundamental because they involve a misunderstanding of either the real problem or the nature of LP models. Category *C* errors are subtler. Generally, a model of a real situation involves some approximation (e.g., many products are aggregated together into a single macro-product, the days of a week are lumped together, or costs that are not quite proportional to volume are nevertheless treated as linear). Avoiding category *C* errors requires skill in identifying which approximations can be tolerated.

With regard to category *A* errors, if the user is fortunate, category *A* errors will manifest themselves by causing solutions that are obviously incorrect.

Errors of formulation are more difficult to discuss because they are of many forms. Doing what we call dimensional analysis can frequently expose the kinds of errors made by a novice. Anyone who has taken a physics or chemistry course would know it as “checking your units.” Let us illustrate by considering an example.

A distributor of toys is analyzing his strategy for assembling Tinkertoy sets for the upcoming holiday season. He assembles two kinds of sets. The “Big” set is composed of 60 sticks and 30 connectors, while the “Tot” set is composed of 30 sticks and 20 connectors. An important factor is, for this season, he has a supply of only 60,000 connectors and 93,000 sticks. He will be able to sell all that he assembles of either set. The profit contributions are \$5.5 and \$3.5 per set, respectively, for Big and Tot. How much should he sell of each set to maximize profit?

The distributor developed the following formulation. Define:

- B = number of Big sets to assemble;
- T = number of Tot sets to assemble;
- S = number of sticks actually used;
- C = number of connectors actually used.

```

MAX = 5.5 * B + 3.5 * T;
B - 30 * C - 60 * S = 0;
T - 20 * C - 30 * S = 0;
C <= 60000;
S <= 93000;

```

Notice the first two constraints are equivalent to:

$$\begin{aligned}B &= 30C + 60S \\T &= 20C + 30S\end{aligned}$$

Do you agree with the formulation? If so, you should analyze its solution below:

Optimal solution found at step:	0	
Objective value:	0.5455500E+08	
Variable	Value	Reduced Cost
B	7380000.	0.0000000
T	3990000.	0.0000000
C	60000.00	0.0000000
S	93000.00	0.0000000
Row	Slack or Surplus	Dual Price
1	0.5455500E+08	1.000000
2	0.0000000	5.500000
3	0.0000000	3.500000
4	0.0000000	235.0000
5	0.0000000	435.0000

There is a hint that the formulation is incorrect because the solution is able to magically produce almost four million Tot sets from only 100,000 sticks.

The mistake that was made is a very common one for newcomers to LP, namely, trying to describe the features of an activity by a constraint. A constraint can always be thought of as a statement that the usage of some item must be less-than-or-equal-to the sources of the item. The last two constraints have this characteristic, but the first two do not.

If one analyzes the dimensions of the components of the first two constraints, one can see there is trouble. The dimensions (or “units”) for the first constraint are:

Term	Units
B	Big sets
30 C	30 [connectors/(Big set)] × connectors
60 S	60 [sticks/(Big set)] × sticks

Clearly, they have different units, but if you are adding items together, they must have the same units. It is elementary that you cannot add apples and oranges. The units of all components of a constraint must be the same.

If one first formulates a problem in words and then converts it to the algebraic form in LINGO, one frequently avoids the above kind of error. In words, we wish to:

Maximize profit contribution
 Subject to:
 Usage of connectors ≤ sources of connectors
 Usage of sticks ≤ sources of sticks

Converted to algebraic form in LINGO, it is:

```
MAX = 5.5 * B + 3.5 * T;
      30 * B + 20 * T <= 60000;
      60 * B + 30 * T <= 93000;
```

The units of the components of the constraint $30 B + 20 T \leq 60,000$ are:

Term	Units
$30 B$	30 [connectors/(Big set)] \times (Big set) = 30 connectors
$20 T$	20 [connectors/(Tot set)] \times (Tot set) = 20 connectors
60,000	60,000 connectors available

Thus, all the terms have the same units of “connectors”. Solving the problem, we obtain the sensible solution:

Optimal solution found at step:	0
Objective value:	10550.00
Variable	Value Reduced Cost
B	200.0000 0.0000000
T	2700.000 0.0000000
Row	Slack or Surplus Dual Price
1	10550.00 1.000000
2	0.0000000 0.1500000
3	0.0000000 0.1666667E-01

4.7 The Nonsimultaneity Error

It must be stressed that all the constraints in an LP formulation apply simultaneously. A combination of activity levels must be found that simultaneously satisfies all the constraints. The constraints do not apply in an either/or fashion, although we might like them to be so interpreted. As an example, suppose we denote by B the batch size for a production run of footwear. A reasonable policy might be, if a production run is made, at least two dozen units should be made. Thus, B will be either zero or some number greater-than-or-equal-to 24. There might be a temptation to state this policy by writing the two constraints:

$$\begin{aligned} B &\leq 0 \\ B &\geq 24. \end{aligned}$$

The desire is that exactly one of these constraints be satisfied. If these two constraints are part of an LP formulation, the computer will reject such a formulation with a curt remark to the effect that no feasible solution exists. There is no unique value for B that is simultaneously less-than-or-equal-to zero and greater-than-or-equal-to 24.

If such either/or constraints are important, then one must resort to integer programming. Such formulations will be discussed in a later section.

4.8 Debugging a Model

LINGO has long had a “Debug” command, see Schrage(1989), that may be helpful in finding formulation errors in models that are infeasible or unbounded. Consider the following model.

```
MAX = 2*X + 3*Y;
[CON1] 2*X + Y <= 12;
[CON2]   X + Y >= 25;
[CON3]   X + 3*Y <= 11;
```

If you try to solve the model, it will be reported as infeasible. Typically, an infeasible or unbounded model contains one or more errors. For an infeasible model, the Debug command (click on: LINGO | Debug) will identify a smallest set of constraints that are infeasible, i.e., cannot all be satisfied. If the infeasibility is due to an error, this set of constraints must contain an error. The Debug report for the above model, if the output level is set to “verbose”, is as follows:

Constraints and bounds that cause an infeasibility:

Sufficient Rows:

(Dropping any sufficient row will make the model feasible.)
[CON2] $X + Y \geq 25$;

Necessary Rows:

(If none of the necessary and sufficient rows are dropped,
then the model remains infeasible.)
[CON1] $2 * X + Y \leq 12$;

Necessary Variable Bounds:

(If none of the necessary and sufficient bounds are dropped,
then the model remains infeasible.)
 $X \geq 0$

The report implies that if you drop the constraint $X + Y \geq 25$, then the model will become feasible. As long as all of the constraints $X+Y \geq 25$, $2*X + Y \leq 12$, and $X \geq 0$ are retained, the model will remain infeasible. Such a set of constraints is sometimes referred to as an “Irreducible Infeasible Set”, or IIS, for short. For more discussion on infeasibility analysis, see Chinneck(2008). Similar debugging analysis is available for unbounded linear programs.

4.9 Problems

- The Tiny Timber Company wants to utilize best the wood resources in one of its forest regions. Within this region, there is a sawmill and a plywood mill. Thus, timber can be converted to lumber or plywood.

Producing a marketable mix of 1000 board feet of lumber products requires 1000 board feet of spruce and 4000 board feet of Douglas fir. Producing 1000 square feet of plywood requires 2000 board feet of spruce and 4000 board feet of Douglas fir. This region has available 32,000 board feet of spruce and 72,000 board feet of Douglas fir.

Sales commitments require at least 5000 board feet of lumber and 12,000 square feet of plywood be produced during the planning period. The profit contributions are \$45 per 1000 board feet of lumber products and \$60 per 1000 square feet of plywood. Let L be the amount (in 1000 board feet) of lumber produced and let P be the amount (in 1000 square feet) of plywood produced. Express the problem as a linear programming model.

- Shmuzzles, Inc., is a struggling toy company that hopes to make it big this year. It makes three fundamental toys: the Shmacrobat, the Shlameleon, and the JigSaw Shmuzzle. Shmuzzles is trying to unload its current inventories through airline in-flight magazines by packaging these three toys in two different size kits, the Dilettante Shmuzzler kit and the Advanced Shmuzzler kit. It's \$29.95 for the Dilettante, whereas the Advanced sells for \$39.95. The compositions of these two kits are:

Dilettante = 6 Shmacrobats plus 10 Shlameleons plus 1 Jig Saw

Advanced = 8 Shmacrobats plus 18 Shlameleons plus 2 Jig Saws

Current inventory levels are: 6,000 Shmacrobats, 15,000 Shlameleons, and 1,500 JigSaws. Formulate a model for helping Shmuzzles, Inc., maximize its profits.

- A standard problem encountered by many firms when introducing new products is the "phase-out" problem. Given the components for products that are being phased out, the question is: what amounts of the phased out products should be built so as to most profitably use the available inventory. The following illustrates. The R. R. Bean Company produces, packages, and distributes freeze-dried food for the camping and outdoor sportsman market. R. R. Bean is ready to introduce a new line of products based on a new drying technology that produces a higher quality, tastier food. The basic ingredients of the current (about to be discontinued) line are dried fruit, dried meat and dried vegetables. There are two products in the current (to be phased out) line: the "Weekender" and the "ExpeditionPak". In its "close-out" catalog, the selling prices of the two products are \$3.80 and \$7.00 per package, respectively. Handling and shipping costs are \$1.50 per package for each package. It is R. R. Bean's long standing practice to include shipping and handling at no charge. The "Weekender" package consists of 3 ounces of dried fruit, 7 ounces of dried meat, and 2 ounces of dried vegetables. The makeup of the "ExpeditionPak" package is 5 ounces of dried fruit, 18 ounces of dried meat, and 5 ounces of dried vegetables. R. R. Bean would like to deplete, as most profitably as possible, its inventories of "old technology" fruit, meat, and vegetables before introducing the new line. The current inventories are 10,000 ounces, 25,000 ounces, and 12,000 ounces respectively of fruit, meat, and vegetables. The book values of these inventories are \$2000, \$2500, and \$1800. Any leftover inventory will be given to the local animal shelter at no cost or benefit to R. R. Bean. The prices in the catalog are such that R. R. Bean is confident that it can sell all that it makes of the two products. Formulate and solve an LP that

should be useful in telling R.R. Bean how many “Weekender” and “Expedition Pak” packages should be mixed to maximize profits from its current inventories.

4. Quart Industries produces a variety of bottled food products at its various plants. At its Americus plant, it produces two products, peanut butter and apple butter. There are two scarce resources at this plant: packaging capacity and sterilization capacity. Both have a capacity of 40 hours per week. Production of 1000 jars of peanut butter requires 4 hours of sterilizer time and 5 hours of packaging time, whereas it takes 6 hours of sterilizer time and 4 hours of packaging time to produce 1000 jars of apple butter. The profit contributions per 1000 jars for the two products are \$1100 and \$1300, respectively. Apple butter preparation requires a boil-down process best done in batches of at least 5000 jars. Thus, apple butter production during the week should be either 0, or 5000 or more jars. How much should be produced this week of each product?
5. An important skill in model formulation is the ability to enumerate all alternatives. Scott Wilkerson is a scientist-astronaut aboard a seven-day space shuttle mission. In spite of a modest health problem that is aggravated by the zero gravity of space, Scott has been allowed on the mission because of his scientific skills and because a pharmaceutical company has prepared a set of two types of pills for Scott to take each day to alleviate his medical condition. At the beginning of each day Scott is to take exactly one type X pill and exactly one type Y pill. If he deviates from this scheme, it will be life threatening for him and the shuttle will have to be brought down immediately. On the first day of the mission, Scott gets one type X pill out of the X bottle, but in the process of trying to get a pill out of the Y bottle, two come out. He grasps for them immediately with the hand that has the X pill and now he finds he has three pills in his hand. Unfortunately, the X and Y pills are indistinguishable. Both types look exactly like a standard aspirin. There are just enough pills for the full length mission, so none can be discarded. What should Scott do? (Hint: this problem would be inappropriate in the integer programming chapter.)

5

The Sets View of the World

In Normal form, each attribute/field of an entity/record should depend on the entity key, the whole key, and nothing but the key, so help me Codd.

-anonymous

5.1 Introduction

The most powerful feature of LINGO is its ability to model large systems. The key concept that provides this power is the idea of a set of similar objects. When you are modeling situations in real life, there will typically be one or more groups of similar objects. Examples of such groups might be factories, products, time periods, customers, vehicles, employees, etc. LINGO allows you to group similar objects together into *sets*. Once the objects in your model are grouped into sets, you can make single statements in LINGO that apply to all members of a set.

A LINGO model of a large system will typically have three sections: 1) a SETS section, 2) a DATA section, and 3) a model equations section. The SETS section describes the data structures to be used for solving a certain class of problems. The DATA section provides the data to “populate” the data structures. The model equations section describes the relationships between the various pieces of data and our decisions.

5.1.1 Why Use Sets?

In most large models, you will need to express a group of several very similar calculations or constraints. LINGO’s ability to handle sets allows you to express such formulae or constraints efficiently.

For example, preparing a warehouse-shipping model for 100 warehouses would be tedious if you had to write each constraint explicitly (e.g., “Warehouse 1 can ship no more than its present inventory, Warehouse 2 can ship no more than its present inventory, Warehouse 3 can ship no more than its present inventory...” and so on). You would prefer to make a single general statement of the form: “Each warehouse can ship no more than its present inventory”.

5.1.2 What Are Sets?

A set is a group of similar objects. A set might be a list of products, trucks, employees, etc. Each member in the set may have one or more characteristics associated with it (e.g., weight, price/unit, or income). We call these characteristics *attributes*. All members of the same set have the same set of attribute types. Attribute values can be known in advance or unknowns for which LINGO solves. For example, each product in a set of products might have an attribute listing its price. Each truck in a set

of trucks might have a hauling capacity attribute. In addition, each employee in a set of employees might have an attribute specifying salary as well as an attribute listing birth date.

5.1.3 Types of Sets

LINGO recognizes two kinds of sets: *primitive* and *derived*. A primitive set is a set composed only of objects that can't be further reduced.

A derived set is defined from one or more other sets using two operations: a) selection (of a subset), and/or b) Cartesian product (sometimes called a “cross” or a “join”) of two or more other sets. The key concept is that a derived set *derives* its members from other pre-existing sets. For example, we might have the two primitive sets: *WAREHOUSE* and *CUSTOMER*. We might have the derived set called *SHIPLINK*, which consists of every possible combination of a warehouse and a customer. Although the set *SHIPLINK* is derived solely from primitive sets, it is also possible to build derived sets from other derived sets as well.

5.2 The SETS Section of a Model

In a set-based LINGO model, the first section in the model is usually the *SETS section*. A SETS section begins with the keyword `SETS:` (including the colon) and ends with the keyword `ENDSETS`. A model may have no SETS section, a single SETS section, or multiple SETS sections. A SETS section may appear almost anywhere in a model. The major restriction is that you must define a set and its attributes before they are referenced in the model's constraints.

5.2.1 Defining Primitive Sets

To define a primitive set in a SETS section, you specify:

- ◆ the *name* of the set, and
- ◆ any *attributes* the members of the set may have.

A primitive set definition has the following syntax¹:

`setname:[attribute_list];`

The *setname* is a name you choose. It should be a descriptive name that is easy to remember. The set name must conform to standard LINGO naming conventions: begin with an alphabetic character, followed by up to 31 alphanumeric characters or the underscore (`_`). LINGO does not distinguish between upper and lowercase characters in names.

An example sets declaration is:

```
SETS:
  WAREHOUSE: CAPACITY;
ENDSETS
```

This means that we will be working with one or more warehouses. Each one of them has an attribute called *CAPACITY*. Set members may have zero or more *attributes* specified in the *attribute_list* of the set definition. An *attribute* is some property each member of the set possesses. Attribute names must follow standard naming conventions and be separated by commas.

¹The use of Square brackets indicates that a particular item is optional. In this particular case, a primitive set's *member_list* and *attribute_list* are optional.

For illustration, suppose our warehouses had additional attributes related to their location and the number of loading docks. These additional attributes could be added to the attribute list of the set declaration as:

```
WAREHOUSE: CAPACITY, LOCATION, DOCKS;
```

5.2.2 Defining Derived Sets

To define a derived set, you specify:

- ◆ the *name* of the set,
- ◆ its *parent sets*,
- ◆ optionally, any *attributes* the set members may have.

A derived set definition has the following syntax:

```
set_name (parent_set_list) [membership_filter] [:attribute_list];
```

The *set_name* is a standard LINGO name you choose to name the set. The optional *membership_filter* may place a general condition on membership in the set.

The *parent_set_list* is a list of previously defined sets, separated by commas. LINGO constructs all the combinations of members from each of the parent sets to create the members of the derived set. As an example, consider the following SETS section:

```
SETS:
  PRODUCT ;
  MACHINE ;
  WEEK;
  ALLOWED( PRODUCT, MACHINE, WEEK) : VOLUME;
ENDSETS
```

Sets *PRODUCT*, *MACHINE*, and *WEEK* are primitive sets, while *ALLOWED* is derived from parent sets *PRODUCT*, *MACHINE*, and *WEEK*. Unless specified otherwise, the set *ALLOWED* will have one member for every combination of *PRODUCT*, *MACHINE*, and *WEEK*. The attribute *VOLUME* might be used to specify how much of each product is produced on each machine in each week. A derived set that contains all possible combinations of members is referred to as being a *dense* set. When a set declaration includes a *membership_filter* or if the members of the derived set are given explicitly in a DATA section, then we say the set is *sparse*.

Summarizing, a derived set's members may be constructed by either:

- ◆ an explicit member list in a DATA section,
- ◆ a membership filter, or
- ◆ implicitly dense by saying nothing about the membership of the derived set.

Specification of an explicit membership list for a derived set in a DATA section will be illustrated in the next section of the text.

If you have a large, sparse set, explicitly listing all members can become cumbersome. Fortunately, in many sparse sets, the members all satisfy some condition that differentiates them from the non-members. If you can specify this condition, you can save yourself a lot of typing. This is exactly how the membership filter method works. Using the membership filter method of defining a derived set's *member_list* involves specifying a logical condition that each potential set member must satisfy for inclusion in the set. You can look at the logical condition as a *filter* that filters out potential members who don't measure up to some criteria.

As an example of a membership filter, suppose you have already defined a set called *TRUCKS* and each truck has an attribute called *CAPACITY*. You would like to derive a subset from *TRUCKS* that contains only those trucks capable of hauling big loads. You could use an explicit member list and explicitly enter each of the trucks that can carry heavy loads. However, why do all that work when you could use a membership filter as follows:

```
HEAVY_DUTY( TRUCKS ) | CAPACITY( &1 ) #GT# 50000;
```

We have named the set *HEAVY_DUTY* and have derived it from the parent set *TRUCKS*. The vertical bar character (|) is used to mark the beginning of a membership filter. The membership filter allows only those trucks that have a hauling capacity (*CAPACITY(&1)*) greater than (#GT#) 50,000 into the *HEAVY_DUTY* set. The &1 symbol in the filter is known as a *set index placeholder*. When building a derived set that uses a membership filter, LINGO generates all the combinations of parent set members. Each combination is then "plugged" into the membership condition to see if it passes the test. The first parent set's value is plugged into &1, the second into &2, and so on. In this example, we have only one parent set (*TRUCKS*), so &2 would not have made sense. The symbol #GT# is a *logical operator* and means "greater than". Other logical operators recognized by LINGO include:

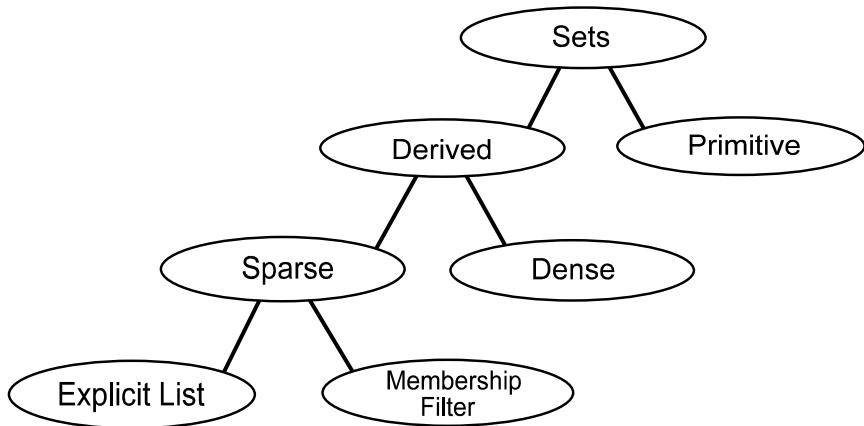
- ◆ #EQ# equal
- ◆ #NE# not equal
- ◆ #GE# greater-than-or-equal-to
- ◆ #LT# less than
- ◆ #LE# less-than-or-equal-to

5.2.3 Summary

LINGO recognizes two types of sets - *primitive* and *derived*. Primitive sets are the fundamental objects in a model and can't be broken down into smaller components. Derived sets, on the other hand, are created from other component sets. These component sets are referred to as the parents of the derived set and may be either primitive or derived.

A derived set can be either *sparse* or *dense*. Dense sets contain all combinations of the parent set members (sometimes this is also referred to as the *Cartesian product* or *cross* of the parent sets). Sparse sets contain only a subset of the cross of the parent sets. These may be defined by two methods - *explicit listing* or *membership filter*. The explicit listing method involves listing the members of the sparse set in a DATA section. The membership filter method allows you to specify the sparse set members compactly using a logical condition, which all members must satisfy. The relationships amongst the various set types are illustrated in Figure 5.1 below.

Figure 5.1 Types of Sets



5.3 The DATA Section

A SETS section describes the structure of the data for a particular class of problems. A DATA section provides the data to create a specific instance of this class of problems. The DATA section allows you to isolate things that are likely to change from week to week. This is a useful practice in that it leads to easier model maintenance and makes a model easier to scale up or down in dimension.

We find it useful to partition a LINGO model of a large system into three distinct sections: a) the SETS section, b) the DATA section, and c) the model equations section. The developer of a model has to understand all three sections. However, if the developer has done a good job of partitioning the model into the aforementioned sections, the day-to-day user may only need to be familiar with the DATA section.

Similar to the SETS section, the DATA section begins with the keyword `DATA:` (including the colon) and ends with the keyword `ENDDATA`. In the DATA section, you place statements to initialize either the attributes of the member of a set you defined in a SETS section or even the set members. These expressions have the syntax:

`attribute_list = value_list;`

or

`set_name = member_list;`

The *attribute_list* contains the names of the attributes you want to initialize, optionally separated by commas. If there is more than one attribute name on the left-hand side of the statement, then all attributes must be associated with the same set. The *value_list* contains the values you want to assign to the attributes in the *attribute_list*, optionally separated by commas. Consider the following example:

```
SETS:
    SET1: X, Y;
ENDSETS
DATA:
SET1 = M1, M2, M3;
X =   1   2   3;
Y =   4   5   6;
ENDDATA
```

We have two attributes, *X* and *Y*, defined on the set *SET1*. The three values of *X* are set to 1, 2, and 3, while *Y* is set to 4, 5, and 6. We could have also used the following compound data statement to the same end:

```
SETS:
    SET1: X, Y;
ENDSETS
DATA:
SET1 X Y =
M1   1   4
M2   2   5
M3   3   6;
ENDDATA
```

Looking at this example, you might imagine *X* would be assigned the values 1, 4, and 2, since they are first in the values list, rather than the true values of 1, 2, and 3. When LINGO reads a data statement's value list, it assigns the first *n* values to the first position of each of the *n* attributes in the attribute list, the second *n* values to the second position of each of the *n* attributes, and so on. In other words, LINGO is expecting the input data in column form rather than row form.

The DATA section can also be used for specifying members of a derived set. The following illustrates both how to specify set membership in a DATA section and how to specify a sparse derived set. This example also specifies values for the *VOLUME* attribute, although that is not required:

```
SETS:
    PRODUCT ;
    MACHINE ;
    WEEK ;
    ALLOWED( PRODUCT, MACHINE, WEEK) : VOLUME;
ENDSETS
DATA:
PRODUCT = A B;
MACHINE = M N;
WEEK = 1..2;
ALLOWED, VOLUME =
A M 1  20.5
A N 2  31.3
B N 1  15.8;
ENDDATA
```

The *ALLOWED* set does not have the full complement of eight members. Instead, *ALLOWED* is just the three member sparse set:

$(A, M, 1), (A, N, 2)$, and $(B, N, 1)$.

LINGO recognizes a number of standard sets. For example, if you declare in a DATA section:

PRODUCT = 1..5;

then the members of the *PRODUCT* set will in fact be 1, 2, 3, 4, and 5. If you declare:

PERIOD = Feb..May;

then the members of the *PERIOD* set will in fact be Feb, Mar, Apr, and May. Other examples of inferred sets include mon..sun and thing1..thing12.

If an attribute is not referenced in a DATA section, then it is by default a decision variable. LINGO may set such an attribute to whatever value is consistent with the statements in the model equations section.

This section gave you a brief introduction to the use of the DATA section. Data do not have to actually reside in the DATA section as shown in these examples. In fact, a DATA section can have OLE links to Excel, ODBC links to databases, and connections to other spreadsheet and text based data files. Examples are given later in this chapter.

Note, when LINGO constructs the derived set, it is the right-most parent set that is incremented the fastest.

5.4 Set Looping Functions

In the model equations section of a model, we state the relationships among various attributes. Any statements not in a SETS or DATA section are by default in the model equations section. The power of set based modeling comes from the ability to apply an operation to all members of a set using a single statement. The functions in LINGO that allow you to do this are called *set looping functions*. If your models do not make use of one or more set looping functions, you are missing out on the power of set based modeling and, even worse, you're probably working too hard!

Set looping functions allow you to iterate through all the members of a set to perform some operation. There are four set looping functions in LINGO. The names of the functions and their uses are:

Function	Function's Use
@FOR	Used to generate constraints over members of a set.
@SUM	Computes the sum of an expression over all members of a set.
@MIN	Computes the minimum of an expression over all members of a set.
@MAX	Computes the maximum of an expression over all members of a set.

The syntax for a set looping function is:

$\text{@loop_function} (\text{setname} [(\text{set_index_list}) \\ [| \text{conditional_qualifier}]] : \text{expression_list});$

The `@loop_function` symbol corresponds to one of the four set looping functions listed in the table above. The `setname` is the name of the set over which you want to loop. The `set_index_list` is optional and is used to create a list of indices each of which correspond to one of the parent, primitive sets that form the set specified by `setname`. As LINGO loops through the members of the set `setname`, it will set the values of the indices in the `set_index_list` to correspond to the current member of the set `setname`. The `conditional_qualifier` is an optional filter and may be used to limit the scope of the set looping function. When LINGO is looping over each member of `setname`, it evaluates the `conditional_qualifier`. If the `conditional_qualifier` evaluates to true, then the `expression_list` of the `@loop_function` is performed for the set member. Otherwise, it is skipped. The `expression_list` is a list of expressions to be applied to each member of the set `setname`. When using the `@FOR` function, the expression list may contain multiple expressions that are separated by semicolons. These expressions will be added as constraints to the model. When using the remaining three set looping functions (`@SUM`, `@MAX`, and `@MIN`), the expression list must contain only one expression. If the `set_index_list` is omitted, all attributes referenced in the `expression_list` must be defined on the set `setname`.

5.4.1 @SUM Set Looping Function

In this example, we will construct several summation expressions using the `@SUM` function in order to illustrate the features of set looping functions in general and the `@SUM` function in particular.

Consider the model:

```
SETS:
    SET_A : X;
ENDSETS
DATA:
    SET_A = A1 A2 A3 A4 A5;
    X = 5 1 3 4 6;
ENDDATA
X_SUM = @SUM( SET_A( J) : X( J));
```

LINGO evaluates the `@SUM` function by first initializing an internal accumulator to zero. LINGO then begins looping over the members in `SET_A`. You can think of `J` as a pronoun. The index variable `J` is first set to the first member of `SET_A` (i.e., `A1`) and $X(A1)$ is then added to the accumulator. Then `J` is set to the second element and this process continues until all values of `X` have been added to the accumulator. The value of the sum is then stored into the variable `X_SUM`.

Since all the attributes in our expression list (in this case, only `X` appears in the expression list) are defined on the index set (`SET_A`), we could have alternatively written our sum as:

```
X_SUM = @SUM( SET_A: X);
```

In this case, we have dropped the superfluous index set list and the index on `X`. When an expression uses this shorthand, we say the index list is *implied*. Implied index lists are not allowed when attributes in the expression list have different parent sets.

Next, suppose we want to sum the first three elements of the attribute `X`. We can use a conditional qualifier on the set index to accomplish this as follows:

```
X3_SUM = @SUM( SET_A( J) | J #LE# 3: X( J));
```

The `#LE#` symbol is called a *logical operator*. This operator compares the operand on the left (`J`) with the one on the right (3) and returns *true* if the left operand is less-than-or-equal-to the one on the right. Otherwise, it returns *false*. Therefore, this time, when LINGO computes the sum, it plugs the set

index variable J into the conditional qualifier $J \#LE\# 3$. If the conditional qualifier evaluates to true, $X(J)$ will be added to the sum. The end result is that LINGO sums up the first three terms in X , omitting the fourth and fifth terms, for a total sum of 9.

Before leaving this example, one subtle aspect to note in this last sum expression is the value that the set index J is returning. Note we are comparing the set index variable to the quantity 3 in the conditional qualifier $J \#LE\# 3$. In order for this to be meaningful, J must represent a numeric value. Since a set index is used to loop over set members, one might imagine a set index is merely a placeholder for the current set member. In a sense, this is true. However, what set indexes *really* return is the *index* of the current set member in its parent primitive set. The index returned is *one-based*. In other words, the value 1 is returned when indexing the first set member, 2 when indexing the second, and so on. Given that set indices return a numeric value, they may be used in arithmetic expressions along with other variables in your model.

5.4.2 @MIN and @MAX Set Looping Functions

The @MIN and @MAX functions are used to find the minimum and maximum of an expression over members of a set. Again, consider the model:

```
SETS:
  SET_A : X;
ENDSETS
DATA:
  SET_A = A1 A2 A3 A4 A5;
  X = 5 1 3 4 6;
ENDDATA
```

To find the minimum and maximum values of X , all one need do is add the two expressions:

```
THE_MIN_OF_X = @MIN(SET_A(J):X(J));
THE_MAX_OF_X = @MAX(SET_A(J):X(J));
```

As with the @SUM example above, we can use an implied index list since the attributes are defined on the index set. Using implied indexing, we can recast our expressions as:

```
THE_MIN_OF_X = @MIN(SET_A:X);
THE_MAX_OF_X = @MAX(SET_A:X);
```

In either case, when we solve this model, LINGO returns the expected minimum and maximum values of X :

Variable	Value
THE_MIN_OF_X	1.000000
THE_MAX_OF_X	6.000000

For illustration purposes, suppose we had just wanted to compute the minimum and maximum values of the first three elements of X . As with the @SUM example, all we need do is add the conditional qualifier $J \#LE\# 3$. We then have:

```
THE_MIN_OF_X_3 = @MIN(SET_A(J) | J #LE# 3:X(J));
THE_MAX_OF_X_3 = @MAX(SET_A(J) | J #LE# 3:X(J));
```

with solution:

Variable	Value
THE_MIN_OF_X_3	1.000000
THE_MAX_OF_X_3	5.000000

5.4.3 @FOR Set Looping Function

The **@FOR** function is used to generate constraints across members of a set. Whereas scalar based modeling languages require you to explicitly enter each constraint, the **@FOR** function allows you to enter a constraint just once and LINGO does the work of generating an occurrence of the constraint for each of the set members. As such, the **@FOR** statement provides the set based modeler with a very powerful tool.

To illustrate the use of **@FOR**, consider the following:

```
SETS:
    TRUCKS : HAUL;
ENDSETS
DATA:
    TRUCKS = MAC, PETERBILT, FORD, DODGE;
ENDDATA
```

Specifically, we have a primitive set of four trucks with a single attribute titled *HAUL*. If the attribute *HAUL* is used to denote the amount a truck hauls, then we can use the **@FOR** function to limit the amount hauled by each truck to 2,500 pounds with the following expression:

```
@FOR( TRUCKS( T) : HAUL( T) <= 2500);
```

In this case, it might be instructive to view the constraints that LINGO generates from our expression. You can do this by using the *LINGO | Generate* command under Windows or by using the *GENERATE* command on other platforms. Running this command, we find that LINGO generates the following four constraints:

```
HAUL( MAC) <= 2500;
HAUL( PETERBILT) <= 2500;
HAUL( FORD) <= 2500;
HAUL( DODGE) <= 2500;
```

As we anticipated, LINGO generated one constraint for each truck in the set to limit them to a load of 2,500 pounds.

Here is a model that uses an **@FOR** statement (listed in bold) to compute the reciprocal of any five numbers placed into the *GPM* attribute:

```
SETS:
    OBJECT: GPM, MPG;
ENDSETS
DATA:
    OBJECT =     A      B      C      D      E;
    GPM = .0303 .03571 .04545 .07142 .10;
ENDDATA
@FOR( OBJECT( I):
    MPG( I) = 1 / GPM( I)
);
```

Solving this model gives the following values for the reciprocals:

Variable	Value
MPG(A)	33.00330
MPG(B)	28.00336
MPG(C)	22.00220
MPG(D)	14.00168
MPG(E)	10.00000

Since the reciprocal of zero is not defined, we could put a conditional qualifier on our *@FOR* statement that causes us to skip the reciprocal computation whenever a zero is encountered. The following *@FOR* statement accomplishes this:

```
@FOR( OBJECT( I) | GPM( I) #NE# 0:
    MPG( I) = 1 / GPM( I)
);
```

The conditional qualifier (listed in bold) tests to determine if the *GPM* is not equal (*#NE#*) to zero. If so, the computation proceeds.

This was just a brief introduction to the use of the *@FOR* statement. There will be many additional examples in the sections to follow.

5.4.4 Nested Set Looping Functions

The simple models shown in the previous section use *@FOR* to loop over a single set. In larger models, you may need to loop over a set within another set looping function. When one set looping function is used within the scope of another, we call it *nesting*. LINGO allows nesting.

The following is an example of an *@SUM* loop nested within an *@FOR*:

```
! The demand constraints;
@FOR( VENDORS( J):
    @SUM( WAREHOUSES( I): VOLUME( I, J)) = DEMAND( J);
);
```

Specifically, for each vendor, we sum up the shipments going from all the warehouses to that vendor and set the quantity equal to the vendor's demand.

@SUM, *@MAX*, and *@MIN* can be nested within any set looping function. *@FOR* functions, on the other hand, may only be nested within other *@FOR* functions.

5.5 Set Based Modeling Examples

Recall, four types of sets can be created in LINGO:

- ◆ primitive,
- ◆ dense derived,
- ◆ sparse derived - explicit list, and
- ◆ sparse derived - membership filter.

This section will help develop your talents for set based modeling by building and discussing four models. Each of these four models will introduce one of the set types listed above.

5.5.1 Primitive Set Example

The following staff scheduling model illustrates the use of a primitive set. This model may be found in the *SAMPLES* subdirectory off the main LINGO directory under the name *STAFFDEM.LNG*.

The Problem

Suppose you run the popular Pluto Dog's hot dog stand that is open seven days a week. You hire employees to work a five-day workweek with two consecutive days off. Each employee receives the same weekly salary. Some days of the week are busier than others and, based on past experience, you know how many workers are required on a given day of the week. In particular, your forecast calls for these staffing requirements:

Day	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Staff Req'd	20	16	13	16	19	14	12

You need to determine how many employees to start on each day of the week in order to minimize the total number of required employees, while still meeting or exceeding staffing requirements each day of the week.

The Formulation

The first question to consider when building a set based model is, "What are the relevant sets and their attributes?". In this model, we have a single primitive set, the days of the week. We will be concerned with two attributes of the *DAYS* set. The first is the number of staff required on each day. The second is the decision variable of the number of staff to start on each day. If we call these attributes *REQUIRED* and *START*, then we might write the SETS section and DATA sections as:

```

SETS:
  DAYS : REQUIRED, START;
ENDSETS
DATA:
  DAYS = MON TUE WED THU FRI SAT SUN;
  REQUIRED = 20 16 13 16 19 14 12;
ENDDATA

```

We are now at the point where we can begin entering the model's mathematical relations (i.e., the objective and constraints). Let's begin by writing the objective: minimize the total number of employees we start during the week. In standard mathematical notation, we might write:

$$\text{Minimize: } \sum_i START_i$$

The equivalent LINGO statement is very similar. Substitute "MIN=" for "Minimize:" and "@SUM(DAYS(I):" for \sum_i and we have:

```
MIN = @SUM( DAYS( I): START( I));
```

Now, all that's left is to deduce the constraints. There is only one set of constraints in this model. Namely, we must have enough staff on duty each day to meet or exceed staffing requirements. In words, what we want is:

for each day: Staff on duty today \geq Staff required today,

The right-hand side of this expression, *Staff required today*, is given. It is simply the quantity *REQUIRED(I)*. The left-hand side, *Staff on duty today* takes a little thought. Given that all employees are on a five-day on/two day off schedule, the number of employees working today is:

$$\begin{aligned} \text{Number working today} &= \text{Number starting today} + \\ &\text{Number starting 1 day ago} + \text{Number starting 2 days ago} + \\ &\text{Number starting 3 days ago} + \text{Number starting 4 days ago}. \end{aligned}$$

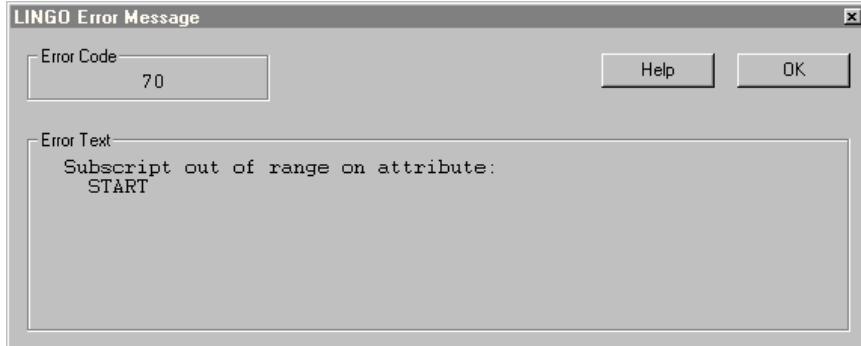
In other words, to compute the number of employees working today, we sum up the number of people starting today plus those starting over the previous four days. The employees starting five and six days back don't count because they are on their days off. Therefore, using mathematical notation, what one might consider doing is adding the constraint:

$$\sum_{i=j-4,j} \text{START}_i \geq \text{REQUIRED}_j, \text{ for } j \in \text{DAYS}$$

Translating into LINGO notation, we can write this as:

```
@FOR( DAYS( J ) :
    @SUM( DAYS( I ) | I #LE# 5: START( J - I + 1 ) )
    >= REQUIRED( J )
);
```

In words, the LINGO statement says, for each day of the week, the sum of the employees starting over the five-day period beginning four days ago and ending today must be greater-than-or-equal-to the required number of staff for the day. This sounds correct, but there is a slight problem. If we try to solve our model with this constraint, we get the error message:



To see why we get this error message, consider what happens on Thursday. Thursday has an index of 4 in our set *DAYS*. As written, the staffing constraint for Thursday will be:

```
START( 4 - 1 + 1) + START( 4 - 2 + 1) +
START( 4 - 3 + 1) + START( 4 - 4 + 1) +
START( 4 - 5 + 1) >= REQUIRED( 4 );
```

Simplifying, we get:

```
START( 4 ) + START( 3 ) +
START( 2 ) + START( 1 ) +
START( 0 ) >= REQUIRED( 4 );
```

It is the $START(0)$ term that is at the root of our problem. $START$ is defined for days 1 through 7. $START(0)$ does not exist. An index of 0 on $START$ is considered "out of range".

What we would like to do is to have any indices less-than-or-equal-to 0, *wrap around* to the end of the week. Specifically, 0 would correspond to Sunday (7), -1 to Saturday (6), and so on. LINGO has a function that does just this, and it is called $@WRAP$.

The $@WRAP$ function takes two arguments - call them $INDEX$ and $LIMIT$. Formally speaking, $@WRAP$ returns J such that $J = INDEX - K \times LIMIT$, where K is an integer such that J is in the interval $[1, LIMIT]$. Informally speaking, $@WRAP$ will subtract or add $LIMIT$ to $INDEX$ until it falls in the range 1 to $LIMIT$, and, therefore, is just what we need to "wrap around" an index in multi-period planning models.

Incorporating the $@WRAP$ function, we get the corrected, final version of our staffing constraint:

```
@FOR( DAYS( J) :
    @SUM( DAYS( I) | I #LE# 5:
        START( @WRAP( J - I + 1, 7))) >= REQUIRED( J)
);
```

The Solution

Below is our staffing model in its entirety:

```
SETS:
    DAYS : REQUIRED, START;
ENDSETS
DATA:
    DAYS = MON TUE WED THU FRI SAT SUN;
    REQUIRED = 20  16  13  16  19  14  12;
ENDDATA
MIN = @SUM( DAYS( I): START( I));
@FOR( DAYS( J):
    @SUM( DAYS( I) | I #LE# 5:
        START( @WRAP( J - I + 1, 7))) >= REQUIRED( J)
);
```

Solving this model, we get the solution report:

Optimal solution found at step:	8	
Objective value:	22.00000	
Variable	Value	Reduced Cost
REQUIRED(MON)	20.00000	0.0000000
REQUIRED(TUE)	16.00000	0.0000000
REQUIRED(WED)	13.00000	0.0000000
REQUIRED(THU)	16.00000	0.0000000
REQUIRED(FRI)	19.00000	0.0000000
REQUIRED(SAT)	14.00000	0.0000000
REQUIRED(SUN)	12.00000	0.0000000
START(MON)	8.00000	0.0000000
START(TUE)	2.00000	0.0000000
START(WED)	0.00000	0.0000000
START(THU)	6.00000	0.0000000
START(FRI)	3.00000	0.0000000
START(SAT)	3.00000	0.0000000
START(SUN)	0.00000	0.0000000

Row	Slack or Surplus	Dual Price
1	22.00000	1.000000
2	0.0000000	-0.2000000
3	0.0000000	-0.2000000
4	0.0000000	-0.2000000
5	0.0000000	-0.2000000
6	0.0000000	-0.2000000
7	0.0000000	-0.2000000
8	0.0000000	-0.2000000

The objective value of 22 means we need to hire 22 workers.

We start our workers according to the schedule:

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Start	8	2	0	6	3	3	0

If we look at the surpluses on our staffing requirement rows (rows 2 - 7), we see the slack values are 0 on all of the days. This means there are no extra workers on any day.

5.5.2 Dense Derived Set Example

The following model illustrates the use of a dense derived set in a blending model. This model may be found in the *SAMPLES* subdirectory off the main LINGO directory under the name *CHESS.LNG*.

The Problem

The Chess Snackfoods Co. markets four brands of mixed nuts. The four brands of nuts are called the *Pawn*, *Knight*, *Bishop*, and *King*. Each brand contains a specified ratio of peanuts and cashews. The table below lists the number of ounces of the two nuts contained in each pound of each brand and the price at which the company can sell a pound of each brand:

	Pawn	Knight	Bishop	King
Peanuts (oz.)	15	10	6	2
Cashews (oz.)	1	6	10	14
Selling Price (\$/lb.)	2	3	4	5

Chess has contracts with suppliers to receive per day: 750 pounds of peanuts and 250 pounds of cashews. Our problem is to determine the number of pounds of each brand to produce each day to maximize total revenue without exceeding the available supply of nuts.

The Formulation

The primitive sets in this model are the nut types and the brands of mixed nuts. The *NUTS* set has the single attribute *SUPPLY* that is the daily supply of nuts in pounds. The *BRANDS* set has *PRICE* and *PRODUCE* attributes, where *PRICE* stores the selling price of the brands and *PRODUCE* represents the decision variables of how many pounds of each brand to produce each day.

We need one more set, however, in order to input the brand formulas. We need a two dimensional table defined on the nut types and the brands. To do this, we will generate a derived set from the cross of the *NUTS* and *BRANDS* sets. Adding this derived set, we get the complete SETS section:

```
SETS:
  NUTS : SUPPLY;
  BRANDS : PRICE, PRODUCE;
  FORMULA( NUTS, BRANDS) : OUNCES;
ENDSETS
```

We have titled the derived set *FORMULA*, and it has the single attribute *OUNCES*, which will be used to store the ounces of nuts used per pound of each brand. Since we have not specified the members of this derived set, LINGO assumes we want the complete, dense set that includes all pairs of nuts and brands.

Now that our sets are defined, we can move on to building the DATA section. We initialize the three attributes *SUPPLY*, *PRICE*, and *OUNCES* in the DATA section as follows:

```
DATA:
  NUTS = PEANUTS, CASHEWS;
  SUPPLY =    750      250;
  BRANDS = PAWN, KNIGHT, BISHOP, KING;
  PRICE =   2       3       4       5;
  OUNCES =  15      10      6       2 ! (Peanuts);
           1       6      10      14; ! (Cashews);
ENDDATA
```

With the sets and data specified, we can enter our objective function and constraints. The objective function of maximizing total revenue is straightforward:

```
MAX = @SUM( BRANDS( I) : PRICE( I) * PRODUCE( I));
```

Our model has only one class of constraints. Namely, we can't use more nuts than we are supplied with on a daily basis. In words, we would like to ensure that:

For each nut type i, the number of pounds of nut i used must be less-than-or-equal-to the supply of nut i.

We can express this in LINGO as:

```
@FOR( NUTS( I) :
  @SUM( BRANDS( J) :
  OUNCES( I, J) * PRODUCE( J) / 16) <= SUPPLY( I)
);
```

We divide the sum on the left-hand side by 16 to convert from ounces to pounds.

The Solution

Our completed nut-blending model is:

```

SETS:
  NUTS : SUPPLY;
  BRANDS : PRICE, PRODUCE;
  FORMULA( NUTS, BRANDS) : OUNCES;
ENDSETS
DATA:
  NUTS = PEANUTS, CASHEWS;
  SUPPLY =    750      250;
  BRANDS = PAWN, KNIGHT, BISHOP, KING;
  PRICE =    2         3         4         5;
  OUNCES = 15        10        6         2 ! (Peanuts);
                1         6        10       14; ! (Cashews);
ENDDATA
MAX = @SUM( BRANDS( I):
  PRICE( I) * PRODUCE( I));
@FOR( NUTS( I):
  @SUM( BRANDS( J):
    OUNCES( I, J) * PRODUCE(J)/16) <= SUPPLY(I)
  );

```

An abbreviated solution report to the model follows:

Optimal solution found at step: 0		
Objective value: 2692.308		
Variable	Value	Reduced Cost
PRODUCE(PAWN)	769.2308	0.0000000
PRODUCE(KNIGHT)	0.0000000	0.1538461
PRODUCE(BISHOP)	0.0000000	0.7692297E-01
PRODUCE(KING)	230.7692	0.0000000
Row	Slack or Surplus	Dual Price
1	2692.308	1.000000
2	0.0000000	1.769231
3	0.0000000	5.461538

This solution tells us Chess should produce 769.2 pounds of the Pawn mix and 230.8 of the King for total revenue of \$2692.30. The dual prices on the rows indicate Chess should be willing to pay up to \$1.77 for an extra pound of peanuts and \$5.46 for an extra pound of cashews. If, for marketing reasons, Chess decides it must produce at least some of the Knight and Bishop mixes, then the reduced cost figures tell us revenue will decrease by 15.4 cents with the first pound of Knight produced and revenue will decline by 76.9 cents with the first pound of Bishop produced.

5.5.3 Sparse Derived Set Example - Explicit List

In this example, we will introduce the use of a sparse derived set with an explicit listing. When using this method to define a sparse set, we must explicitly list all members of the set. This will usually be some small subset of the dense set resulting from the full Cartesian product of the parent sets.

For our example, we will set up a PERT (*Program Evaluation and Review Technique*) model to determine the *critical path* of tasks in a project involving the roll out of a new product. PERT is a simple, but powerful, technique developed in the 1950s to assist managers in tracking the progress of large projects. Its first official application was to the fleet submarine ballistic missile project, the

so-called Polaris project. According to Craven(2001), PERT was given its name by Admiral William F. Raborn, who played a key role in starting the Polaris project. Raborn had a new bride whose nickname was Pert. In her honor, Raborn directed that the management system that was to monitor the Polaris project be called Pert. In fact, the PERT system proved so successful the Polaris project was completed eighteen months ahead of schedule! PERT is particularly useful at identifying the critical activities within a project, which, if delayed, will delay the project as a whole. These time critical activities are referred to as the critical path of a project. Having such insight into the dynamics of a project goes a long way in guaranteeing it won't get sidetracked and become delayed. PERT, and a closely related technique called CPM (Critical Path Method), continues to be used successfully on a wide range of projects. The formulation for this model is included in the *SAMPLES* subdirectory off the main LINGO directory under the name *PERTD.LNG*.

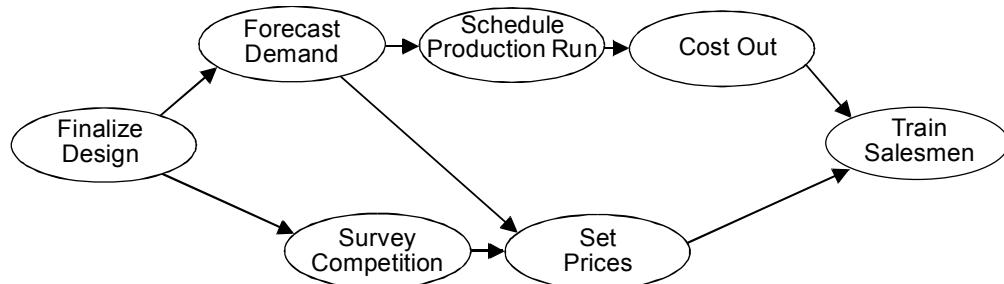
The Problem

Wireless Widgets is about to launch a new product — the Solar Widget. In order to guarantee the launch will occur on time, WW wants to perform a PERT analysis of the tasks leading up to the launch. Doing so will allow them to identify the critical path of tasks that must be completed on time in order to guarantee the Solar Widget's timely introduction. The tasks that must be accomplished before introduction and their anticipated times for completion are listed in the table below:

Task	Weeks
Finalize Design	10
Forecast Demand	14
Survey Competition	3
Set Prices	3
Schedule Production Run	7
Cost Out	4
Train Salesmen	10

Certain tasks must be completed before others can commence. These precedence relations are shown in Figure 5.2:

Figure 5.2 Product Launch Precedence Relations



For instance, the two arrows originating from the *Forecast Demand* node indicate the task must be completed before the *Schedule Production Run* and the *Set Prices* tasks may be started.

Our goal is to construct a PERT model for the Solar Widget's introduction in order to identify the tasks on the critical path.

The Formulation

We will need a primitive set to represent the tasks of the project.

We have associated four attributes with the *TASKS* set. The definitions of the attributes are:

TIME	Time duration to complete the task, given
ES	Earliest possible start time for the task, to be computed,
LS	Latest possible start time for the task, to be computed
SLACK	Difference between LS and ES for the task, to be computed.

If a task has a 0 slack, it means the task must start on time or the whole project will be delayed. The collection of tasks with 0 slack time constitutes the critical path for the project.

In order to compute the start times for the tasks, we will need to examine the precedence relations. Thus, we will need to input the precedence relations into the model. The precedence relations can be viewed as a list of ordered pairs of tasks. For instance, the fact the *DESIGN* task must be completed before the *FORECAST* task could be represented as the ordered pair (*DESIGN*, *FORECAST*). Creating a two-dimensional derived set on the *TASKS* set will allow us to input the list of precedence relations. Therefore, our DATA section will look as follows:

```
DATA:
  TASKS : TIME, ES, LS, SLACK;
  PRED( TASKS, TASKS );
```

Notice that the *PRED* set has no attributes. Its purpose is only to provide the information about the precedence relationships between tasks.

Next, we can input the task times and precedence pairs in the DATA section thus:

```
DATA:
  TASKS= DESIGN, FORECAST, SURVEY, PRICE, SCHEDULE, COSTOUT, TRAIN;
  TIME =    10,        14,        3,        3,        7,        4,        10;
  PRED =
    DESIGN,FORECAST,
    DESIGN,SURVEY,
    FORECAST,PRICE,
    FORECAST,SCHEDULE,
    SURVEY,PRICE,
    SCHEDULE,COSTOUT,
    PRICE,TRAIN,
    COSTOUT,TRAIN;
ENDDATA
```

Keep in mind that the first member of the *PRED* set is the ordered pair (*DESIGN*, *FORECAST*) and not just the single task *DESIGN*. Therefore, this set has a total of 8 members. Each of which corresponds to an arc in the precedence relations diagram.

The feature to note from this example is that the set *PRED* is a sparse derived set with an explicit listing of members. The set is a subset derived from the cross of the *TASKS* set upon itself. The set is sparse because it contains only 8 out of a possible 49 members found in the complete cross of *TASKS* on *TASKS*. The set has an explicit listing because we have included a listing of the members we want included in the set. Explicitly listing the members of a sparse set may not be convenient in cases where there are thousands of members to select from, but it does make sense whenever set membership conditions are not well-defined and the sparse set size is small relative to the dense alternative.

Now, with our sets and data established, we can turn our attention to building the formulas of the model. We have three attributes to compute: earliest start (*ES*), latest start (*LS*), and slack time (*SLACK*). The trick is computing *ES* and *LS*. Once we have these times, *SLACK* is merely the difference of the two. Let's start by deriving a formula to compute *ES*. A task cannot begin until all its predecessor tasks are completed. Thus, if we find the latest finishing time of all predecessors to a task, then we have also found its earliest start time. Therefore, in words, the earliest start time for task *t* is equal to the maximum of the sum of the earliest start time of the predecessor plus its completion time over all predecessors of task *t*. The corresponding LINGO notation is:

```
@FOR( TASKS( J) | J #GT# 1:
      ES( J) = @MAX( PRED( I, J): ES( I) + TIME( I)) );
```

Note, we skip the computation for task 1 by adding the conditional qualifier *J #GT# 1*. We do this because task 1 has no predecessors. We will give the first task an arbitrary start time of 0 below.

Computing *LS* is similar to *ES*, except we must think backwards. In words, the latest time for task *t* to start is the minimum, over all successor tasks *j*, of *j*'s latest start minus the time to perform task *t*. If task *t* starts any later than this, it will force at least one successor to start later than its latest start time. Converting into LINGO syntax gives:

```
@FOR( TASKS( I) | I #LT# LTASK:
      LS( I) = @MIN( PRED( I, J): LS( J) - TIME( I)) );
```

Here, we omit the computation for the last task, since it has no successor tasks.

Computing slack time is just the difference between *LS* and *ES* and may be written as:

```
@FOR( TASKS( I): SLACK( I) = LS( I) - ES( I));
```

We can set the start time of task 1 to some arbitrary value. For our purposes, we will set it to 0 with the statement:

```
ES( 1) = 0;
```

We have now input formulas for computing the values of all the variables with the exception of the latest start time for the last task. It turns out, if the last project were started any later than its earliest start time, the entire project would be delayed. So, by definition, the latest start time for the last project is equal to its earliest start time. We can express this in LINGO using the equation:

```
LS( 7) = ES( 7);
```

This would work, but it is not a very general way to express the relation. Suppose you were to add some tasks to your model. You'd have to change the 7 in this equation to whatever the new number of tasks was. The whole idea behind LINGO's set based modeling language is the equations in the model should not need changing each time the data change. Expressing the equation in this form violates data independence. Here's a better way to do it:

```
LTASK = @SIZE( TASKS);
LS( LTASK) = ES( LTASK);
```

The *@SIZE* function returns the size of a set. In this case, it will return the value 7, as desired. However, if we changed the number of tasks, *@SIZE* would also return the new, correct value. Thus, we preserve the data independence of our model's structure.

The Solution

The entire PERT formulation and portions of its solution appear below:

```

SETS:
  TASKS : TIME, ES, LS, SLACK;
  PRED( TASKS, TASKS);
ENDSETS
DATA:
  TASKS= DESIGN, FORECAST, SURVEY, PRICE, SCHEDULE, COSTOUT, TRAIN;
  TIME =    10,        14,        3,        3,        7,        4,        10;
  PRED =
    DESIGN,FORECAST,
    DESIGN,SURVEY,
    FORECAST,PRICE,
    FORECAST,SCHEDULE,
    SURVEY,PRICE,
    SCHEDULE,COSTOUT,
    PRICE,TRAIN,
    COSTOUT,TRAIN;
ENDDATA
@FOR( TASKS( J) | J #GT# 1:
  ES( J) = @MAX( PRED( I, J): ES( I) + TIME( I))
  );
@FOR( TASKS( I) | I #LT# LTASK:
  LS( I) = @MIN( PRED( I, J): LS( J) - TIME( I));
  );
@FOR( TASKS( I): SLACK( I) = LS( I) - ES( I));
ES( 1) = 0;
LTASK = @SIZE( TASKS);
LS( LTASK) = ES( LTASK);

```

The interesting part of the solution is:

Variable	Value
LTASK	7.000000
ES(DESIGN)	0.000000
ES(FORECAST)	10.000000
ES(SURVEY)	10.000000
ES(PRICE)	24.000000
ES(SCHEDULE)	24.000000
ES(COSTOUT)	31.000000
ES(TRAIN)	35.000000
LS(DESIGN)	0.000000
LS(FORECAST)	10.000000
LS(SURVEY)	29.000000
LS(PRICE)	32.000000
LS(SCHEDULE)	24.000000
LS(COSTOUT)	31.000000
LS(TRAIN)	35.000000
SLACK(DESIGN)	0.000000
SLACK(FORECAST)	0.000000
SLACK(SURVEY)	19.000000
SLACK(PRICE)	8.000000
SLACK(SCHEDULE)	0.000000

SLACK(COSTOUT)	0.000000
SLACK(TRAIN)	0.000000

The interesting values are the slacks for the tasks. *SURVEY* and *PRICE* have respective slacks of 19 and 8. The start time of either *SURVEY* or *PRICE* (but not both) may be delayed by as much as these slack values without delaying the completion time of the entire project. The tasks *DESIGN*, *FORECAST*, *SCHEDULE*, *COSTOUT*, and *TRAIN*, on the other hand, have 0 slack. These tasks constitute the critical path. If any of their start times are delayed, the entire project will be delayed. Management will want to pay close attention to these critical path activities to be sure they start on time and complete within the allotted time. Finally, the *ES(TRAIN)* value of 35 tells us the estimated time to the start of the roll out of the new Solar Widget will be 45 weeks: 35 weeks to get to the start of training, plus 10 weeks to complete training.

5.5.4 A Sparse Derived Set Using a Membership Filter

In this example, we introduce the use of a sparse derived set with a membership filter. Using a membership filter is the third method for defining a derived set. When you define a set using this method, you specify a logical condition each member of the set must satisfy. This condition is used to filter out members that don't satisfy the membership condition.

For our example, we will formulate a *matching* problem. In a matching problem, there are N objects we want to match into pairs at minimum cost. Sometimes this is known as the roommate selection problem. It is a problem faced by a university at the beginning of each school year as incoming first year students are assigned to rooms in dormitories. The pair (I,J) is indistinguishable from the pair (J,I) . Therefore, we arbitrarily require I be less than J in the pair. Formally, we require I and J make a set of ordered pairs. In other words, we do not wish to generate redundant ordered pairs of I and J , but only those with I less than J . This requirement that I be less than J will form our membership filter.

The file containing this model may be found in the *SAMPLES* subdirectory off the main LINGO directory under the name *MATCHD.LNG*.

The Problem

Suppose you manage your company's strategic planning department. There are eight analysts in the department. Your department is about to move into a new suite of offices. There are four offices in the new suite and you need to match up your analysts into 4 pairs, so each pair can be assigned to one of the new offices. Based on past observations you know some of the analysts work better together than they do with others. In the interest of departmental peace, you would like to come up with a pairing of analysts that results in minimal potential conflicts. To this goal, you have come up with a rating system for pairing your analysts. The scale runs from 1 to 10, with a 1 rating for a pair meaning the two get along fantastically, whereas all sharp objects should be removed from the pair's office in anticipation of mayhem for a rating of 10. The ratings appear in the following table:

Analysts	1	2	3	4	5	6	7	8
1	-	9	3	4	2	1	5	6
2	-	-	1	7	3	5	2	1
3	-	-	-	4	4	2	9	2
4	-	-	-	-	1	5	5	2
5	-	-	-	-	-	8	7	6
6	-	-	-	-	-	-	2	3
7	-	-	-	-	-	-	-	4

Analysts' Incompatibility Ratings

Since the pairing of analyst I with analyst J is indistinguishable from the pairing of J with I , we have only included the above diagonal elements in the table. Our problem is to find the pairings of analysts that minimizes the sum of the incompatibility ratings of the paired analysts.

The Formulation

The first set of interest in this problem is the set of eight analysts. This primitive set can be written simply as:

ANALYSTS;

The final set we want to construct is a set consisting of all the potential pairings. This will be a derived set we will build by taking the cross of the ANALYST set. As a first pass, we could build the dense derived set:

PAIRS(ANALYSTS, ANALYSTS);

This set, however, would include both $PAIRS(I, J)$ and $PAIRS(J, I)$. Since only one of these pairs is required, the second is wasteful. Furthermore, this set will include "pairs" of the same analyst of the form $PAIRS(I, I)$. As much as each of the analysts might like an office of their own, such a solution is not feasible. The solution is to put a membership filter on our derived set requiring each pair (I,J) in the final set to obey the condition J be greater than I . We do this with the set definition:

PAIRS(ANALYSTS, ANALYSTS) | &2 #GT# &1;

The start of the membership filter is denoted with the vertical bar character ($|$). The $\&1$ and $\&2$ symbols in the filter are known as *set index placeholders*. Set index placeholders are valid only in membership filters. When LINGO constructs the PAIRS set, it generates all combinations in the cross of the ANALYSTS set on itself. Each combination is then "plugged" into the membership filter to see if it passes the test. Specifically, for each pair (I,J) in the cross of set ANALYSTS on itself, I is substituted

into the placeholder $\&I$ and J into $\&2$ and the filter is evaluated. If the filter evaluates to true, (I,J) is added to the pairs set. Viewed in tabular form, this leaves us with just the above diagonal elements of the (I,J) pairing table.

We will also be concerned with two attributes of the *PAIRS* set. First, we will need an attribute that corresponds to the incompatibility rating of the pairings. Second, we will need an attribute to indicate if analyst I is paired with analyst J . We will call these attributes *RATING* and *MATCH*. We append them to the *PAIRS* set definition as follows:

```
PAIRS( ANALYSTS, ANALYSTS ) | &2 #GT# &1: RATING, MATCH;
```

We will simply initialize the *RATING* attribute to the incompatibility ratings listed in the table above using the *DATA* section:

```
DATA:
  ANALYSTS = 1..8;
  RATING =
    9 3 4 2 1 5 6
    1 7 3 5 2 1
    4 4 2 9 2
    1 5 5 2
    8 7 6
    2 3
    4;
ENDDATA
```

We will use the convention of letting $MATCH(I, J)$ be 1 if we pair analyst I with analyst J , otherwise 0. As such, the *MATCH* attribute contains the decision variables for the model.

Our objective is to minimize the sum of the incompatibility ratings of all the final pairings. This is just the inner product on the *RATING* and *MATCH* attributes and is written as:

```
MIN = @SUM( PAIRS( I, J):
  RATING( I, J) * MATCH( I, J));
```

There is just one class of constraints in the model. In words, what we want to do is:

For each analyst, ensure the analyst is paired with exactly one other analyst.

Putting the constraint into LINGO syntax, we get:

```
@FOR( ANALYSTS( I):
  @SUM( PAIRS( J, K) | J #EQ# I #OR# K #EQ# I:
  MATCH( J, K)) = 1
);
```

The feature of interest in this constraint is the conditional qualifier $J \#EQ\# I \#OR\# K \#EQ\# I$ on the *@SUM* function. For each analyst I , we sum up all the *MATCH* variables that contain I and set them equal to 1. In so doing, we guarantee analyst I will be paired up with exactly one other analyst. The conditional qualifier guarantees we only sum up the *MATCH* variables that include I in its pairing.

One other feature is required in this model. We are letting $MATCH(I, J)$ be 1 if we are pairing I with J . Otherwise, it will be 0. Unless specified otherwise, LINGO variables can assume any value from 0 to infinity. Since we want *MATCH* to be restricted to being only 0 or 1, we need to add one other feature to our model. What we need is to apply the *@BIN* variable domain function to the *MATCH* attribute. *Variable domain functions* are used to restrict the values a variable can assume. Unlike constraints, variable domain functions do not add equations to a model. The *@BIN* function

restricts a variable to being *binary* (i.e., 0 or 1). When you have a model that contains binary variables, it is said to be an *integer programming* (IP) model. IP models are much more difficult to solve than models that contain only continuous variables. Carelessly formulated IPs (with several hundred integer variables or more) can literally take *forever* to solve! Thus, you should limit the use of binary variables whenever possible. To apply `@BIN` to all the variables in the *MATCH* attribute, add the `@FOR` expression:

```
@FOR( PAIRS( I, J): @BIN( MATCH( I, J)));
```

The Solution

The entire formulation for our matching example and parts of its solution appears below:

```
SETS:
  ANALYSTS;
  PAIRS( ANALYSTS, ANALYSTS) | &2 #GT# &1:
    RATING, MATCH;
ENDSETS
DATA:
  ANALYSTS = 1..8;
  RATING =
    9  3  4  2  1  5  6
    1  7  3  5  2  1
      4  4  2  9  2
        1  5  5  2
          8  7  6
            2  3
              4;
ENDDATA
MIN = @SUM( PAIRS( I, J):
  RATING( I, J) * MATCH( I, J));
@FOR( ANALYSTS( I):
  @SUM( PAIRS( J, K) | J #EQ# I #OR# K #EQ# I:
    MATCH( J, K)) = 1
  );
@FOR( PAIRS( I, J): @BIN( MATCH( I, J)));
```

A solution is:

Variable	Value
MATCH(1, 2)	0.0000000
MATCH(1, 3)	0.0000000
MATCH(1, 4)	0.0000000
MATCH(1, 5)	0.0000000
MATCH(1, 6)	1.0000000
MATCH(1, 7)	0.0000000
MATCH(1, 8)	0.0000000
MATCH(2, 3)	0.0000000
MATCH(2, 4)	0.0000000
MATCH(2, 5)	0.0000000
MATCH(2, 6)	0.0000000
MATCH(2, 7)	1.0000000
MATCH(2, 8)	0.0000000
MATCH(3, 4)	0.0000000
MATCH(3, 5)	0.0000000
MATCH(3, 6)	0.0000000

MATCH(3, 7)	0.0000000
MATCH(3, 8)	1.0000000
MATCH(4, 5)	1.0000000
MATCH(4, 6)	0.0000000
MATCH(4, 7)	0.0000000
MATCH(4, 8)	0.0000000
MATCH(5, 6)	0.0000000
MATCH(5, 7)	0.0000000
MATCH(5, 8)	0.0000000
MATCH(6, 7)	0.0000000
MATCH(6, 8)	0.0000000
MATCH(7, 8)	0.0000000

Notice from the objective value, the total sum of incompatibility ratings for the optimal pairings is 6. Scanning the *Value* column for 1's, we find the optimal pairings: (1,6), (2,7), (3,8), and (4,5).

5.6 Domain Functions for Variables

Variable domain functions were briefly introduced in this chapter when we used *@BIN* in the previous matching model. Variable domain functions allow one to put restrictions on the values allowed for decision variables. Examples of the four domain functions available are:

```
@BIN( Y);
@GIN( X);
@BND( 100, DELIVER, 250);
@FREE( PROFIT);
```

The statement *@BIN(Y)* restricts the variable *Y* to be a binary variable. That is, it can take on only the values 0 and 1.

The statement *@GIN(X)* restricts the variable *X* to be a general integer variable. That is, it can take on only the values 0, 1, 2, ...

The *@BND()* specification allows one to specify simple upper and lower bounds. The statement *@BND(100, DELIVER, 250)* restricts the variable *DELIVER* to be in the interval [100, 250]. The same effect could be achieved by the slightly more verbose:

```
DELIVER >= 100;
DELIVER <= 250;
```

LINGO, by default, gives a lower bound of zero to every decision variable. The statement *@FREE(PROFIT)* overrides this default lower bound for the variable *PROFIT* and says that (unfortunately) *PROFIT* can take on any value between minus infinity and plus infinity. Each of the domain functions can appear inside *@FOR* loops, just like any other constraint.

5.7 Spreadsheets and LINGO

In this chapter, we have seen how LINGO can be useful for modeling very large problems. The most widely used method for modeling of any sort is undoubtedly spreadsheet models. When is which approach more appropriate?

The major advantages of doing a model in a spreadsheet are:

- Excellent report formatting features available,
- Large audience of people who understand spreadsheets, and
- Good interface capability with other systems such as word processors.

The major advantages of doing a model in LINGO are:

- Flexibility of various kinds.
- Scalability--It is easy to change the size of any set (e.g., add time periods, products, customers, suppliers, transportation modes, etc.) without having to worry about copying or editing formulae. There is no upper limit of 255(as in a spreadsheet) on the number of columns, or 65536 on the number of rows. There is no limit of about 900(as in a spreadsheet) characters in a formula.
- Sparse sets are easily represented.
- Auditability and visibility--It is easy to see the formulae of a LINGO model in complete, comprehensive form. Truly understanding the model formulae underlying a complex spreadsheet is an exercise in detective work.
- Multiple dimensions are easily represented. A spreadsheet handles two dimensions very well, three dimensions somewhat well, and four or more dimensions not very well.

One can get most of the benefits of both by using LINGO in conjunction with spreadsheets. One can place "hooks" in a LINGO model, so it automatically retrieves and inserts data from/to spreadsheets, databases, and ordinary files. Under Microsoft Windows, the hooks used are the OLE (Object Linking and Embedding) and ODBC (Open Database Connectivity) interfaces provided as part of Windows. Using the OLE capability to connect an Excel spreadsheet to a LINGO model requires two steps:

- a) In the spreadsheet, each data area that is to be either a supplier to or a receiver of data from the LINGO model must be given an appropriate range name. This is done in the spreadsheet by highlighting the area of interest with the mouse, and then using the *Insert | Name | Define* command. The most convenient name to give to a range is the same name by which the data are referenced in the LINGO model.
- b) In the LINGO model, each attribute (vector) (e.g., plant capacities) that is to be retrieved from a spreadsheet, must appear in a LINGO DATA section in a statement of the form:

CAPACITY = @OLE('C:\MYDATA.XLS');

Each attribute (e.g., amount to ship) to be sent to a spreadsheet must appear in a LINGO DATA section in a statement of the form:

@OLE('C:\MYDATA.XLS') = AMT_SHIPPED;

If only one spreadsheet is open in Excel, this connection can be simplified. You need only write:

CAPACITY = @OLE();

LINGO will look in the only open spreadsheet for the range called CAPACITY. This “unspecified spreadsheet” feature is very handy if you want to apply the same LINGO model to several different spreadsheet data sets.

This spreadsheet connection can be pushed even further by embedding the LINGO model in the spreadsheet for which it has a data connection. This is handy because the associated LINGO model will always be obviously and immediately available when the spreadsheet is opened. The screen shot below shows a transportation model embedded in a spreadsheet. To the casual user, it looks like a standard spreadsheet with a special solve button.

The screenshot shows a Microsoft Excel spreadsheet titled "Trnasportation LP Using Excel and LINGO". The spreadsheet contains a cost matrix for a transportation problem with three warehouses (Wh1, Wh2, Wh3) and four customers (CustA, CustB, CustC, CustD). The matrix values are:

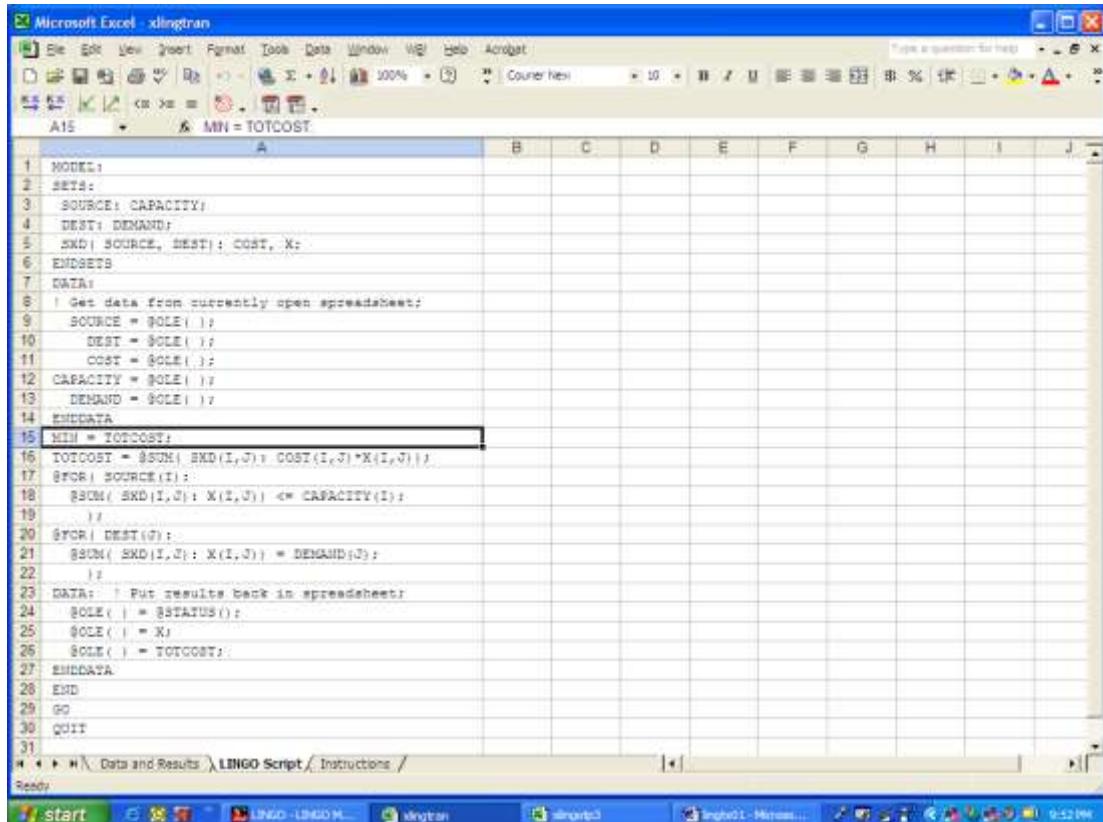
	Capacity	CustA	CustB	CustC	CustD
Demand	4	7	5	3	
Wh1	10	2	6	4	3
Wh2	15	6	6	3	6
Wh3	11	5	3	4	5

A "Solve" button is located in the cell G4. Below the matrix, the total cost is listed as 53, and the recommended shipments are:

	Recommended Shipments
Wh1	4 0 0 3
Wh2	0 6 5 0
Wh3	0 7 0 0

The status bar at the bottom indicates "Data and Results / LINGO Script / Instructions /".

The data and results are stored on the first tab/sheet of the spreadsheet file. Not so obvious is the LINGO model that is stored on another tab in the same spreadsheet (see below). Completely hidden is a small VBA program in the spreadsheet that causes the LINGO model on the second tab to be solved whenever the Solve button is clicked on the first tab. The complete example can be found in the file xlingtran.xls.



The screenshot shows the Microsoft Excel interface with the title bar "Microsoft Excel - xlingtran". The ribbon menu is visible at the top. The active tab is the "LINGO" tab, which contains the LINGO script. The script is as follows:

```

1 MODEL;
2 SETS;
3 SOURCE: CAPACITY;
4 DEST: DEMAND;
5 END(SOURCE, DEST): COST, X;
6 ENDSETS;
7 DATA;
8 ! Get data from currently open spreadsheet;
9 SOURCE = @OLE(1);
10 DEST = @OLE(1);
11 COST = @OLE(1);
12 CAPACITY = @OLE(1);
13 DEMAND = @OLE(1);
14 ENDDATA;
15 MIN = TOTCOST;
16 TOTCOST = $SUM( $ND(I,J): COST(I,J)*X(I,J) );
17 $FOR(1: SOURCE(I):
18   $SUM( $ND(I,J): X(I,J) ) <= CAPACITY(I);
19   );
20 $FOR(1: DEST(J):
21   $SUM( $ND(I,J): X(I,J) ) = DEMAND(J);
22   );
23 DATA; ! Put results back in spreadsheet;
24 @OLE(1) = $STATUS();
25 @OLE(1) = X;
26 @OLE(1) = TOTCOST;
27 ENDDATA;
28 END;
29 GO;
30 QUIT;
31

```

The status bar at the bottom shows the path "Data and Results \ LINGO Script \ Instructions \". The taskbar at the bottom has icons for Start, LINGO-LINK.DLL, xlingtran, and others.

Just as `@OLE()` is used to connect a LINGO model to a spreadsheet and `@ODBC()` is used to connect a LINGO model to most databases that support the SQL interface, the `@TEXT()` statement is available to connect a LINGO model to a simple text file. You can send the value(s) of attribute *X* to a file called "myfile.out" with:

```

DATA;
@TEXT( 'MYFILE.OUT') = X;
ENDDATA

```

The following will send the value of *X* to the screen, along with an explanatory message:

```

@TEXT() = 'The value of X=' , X;

```

Still one more way that LINGO can be incorporated into an application is by way of a subroutine call. A regular computer program, say in C/C++ or Visual Basic, can make a regular call to the LINGO DLL(Dynamic Link Library). The model is passed as a string variable to the LINGO DLL. See the LINGO manual for more details.

5.8 Programming in LINGO

LINGO also has a programming capability(in the sense of computer programming) or looping capability. The main benefits of this are the ability to a) do preprocessing of data to be used in the model, e.g. to do complicated calculations of profit contribution coefficients, b) do postprocessing of solutions to produce customized output rather than the standard LINGO solution report, c) solve 2 or more related models in a single run. The ability to solve multiple models with “one click” makes it easier to do things like i) parametric analysis to show how profit changes as a function of some critical parameter, ii) solve goal programming problems where there is a hierarchy of goals, and iii) build models incrementally by adding variables and/or constraints in an iterative, column-generation fashion.

5.8.1 Building Blocks for Programming

Executable statements are identified by a CALC section:

```
CALC:
    ! Executable statements;
ENDCALC
```

Calculations occur sequentially, from top to bottom in a CALC section except when one of four different “flow control statements: @IFC, @FOR, @WHILE, or @BREAK are encountered. The format of an “If Condition” statement is:

```
@IFC(condition:
    ! Executable Statements;
@ELSE
    ! Executable Statements;
) ;
```

There are two loop control statements, @FOR for looping over a set of known size,

```
@FOR( set | condition:
    ! Executable Statements;
) ;
```

and @WHILE, for looping an initially unknown number of times:

```
@WHILE( condition:
    ! Executable Statements;
) ;
```

One can break out of a loop with:

```
@BREAK
```

An output string can be written with a statement of the form:

```
@WRITE( output_list);
```

where the output_list can be an explicit string, a variable, a variable in a specified format, @FORMAT(), or an end of line character, @NEWLINE(n).

The syntax of the @FORMAT function is:

@FORMAT(math_expression, field_description),
where a field_description is something like “6.2f” for numbers or “7s” for a name.

A model that we want to reference and solve in a CALC section is indicated with the SUBMODEL declaration, e.g.

```
SUBMODEL mymodel:  
    ! Model Statements;  
ENDSUBMODEL
```

We can solve a previously defined submodel in a CALC section with an @SOLVE statement, e.g. :

```
@SOLVE( mymodel);
```

We illustrate programming in LINGO with the computation of an “efficient frontier” for the Astro-Cosmo problem.

```
! Model to compute efficient frontier;  
SUBMODEL ASTROCOSMO:  
    MAX = OBJ;  
    OBJ= 20*A + 30*C;  
        A      <= 60;  
        C <= 50;  
        A + 2*C <= LABORAV;  
ENDSUBMODEL  
  
DATA:  
    ! Number of points to compute in efficient frontier;  
    NPTS = 11;  
    ! Upper limit on labor(lower limit is 0);  
    UPLIM = 200;  
ENDDATA  
  
CALC:  
    ! Set output level to super terse;  
    @SET( 'TERSEO', 2);  
    @WRITE('    Labor      Profit', @NEWLINE(1));  
    ! Loop over points on efficient frontier;  
    i = 0; ! Standard 3 statement loop control construct;  
    @WHILE( i #LT# NPTS:  
        i = i + 1;  
        LABORAV = UPLIM*(i-1) / (NPTS-1);  
  
        ! Solve model with new labor availability;  
        @SOLVE(ASTROCOSMO);
```

```

! Write the objective value, OBJ, in a field of
! 8 characters with 2 digits to the right of decimal point;
@WRITE(" ", @FORMAT(LABORAV, "8.0f"),
      ' ', @FORMAT(OBJ,"8.2f"), @NEWLINE(1));
); ! End @WHILE loop;
ENDCALC

```

This produces the output:

Labor	Profit
0	0.00
20	400.00
40	800.00
60	1200.00
80	1500.00
100	1800.00
120	2100.00
140	2400.00
160	2700.00
180	2700.00
200	2700.00

5.8.2 Generating Graphs and Charts

LINGO can generate about a dozen different chart or graph types such as histograms, pie charts, scatter plots, two dimensional curves, and surface charts. The previous example can be modified to generate a two dimensional curve by adding a small SETS section and modifying the CALC section as follows.

```

SETS:
! Define a grid;
S /1..NPTS/: CX, CY;
ENDSETS

CALC:
! Set output level to super terse;
@SET( 'TERSEO', 2);
! Loop over points on efficient frontier;
i = 0; !Standard 3 statement loop control construct;
@WHILE( i #LT# NPTS:
      i = i + 1;
      LABORAV = UPLIM*(i-1) / (NPTS-1);

      ! Solve model with new labor availability;
      @SOLVE(ASTROCOSMO);

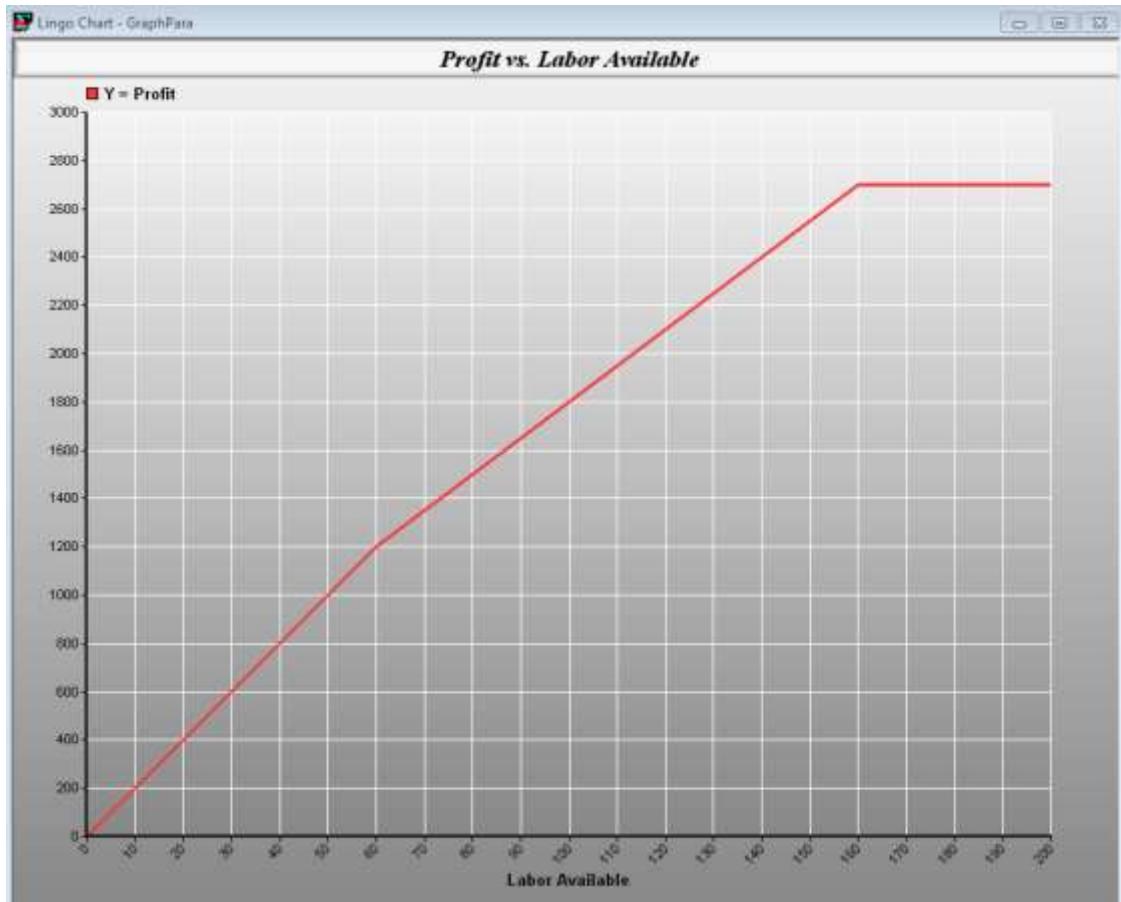
      ! Fill the grid with values at current point;
      CX( i) = LABORAV ;
      CY( i) = OBJ;

); ! End @WHILE loop;

```

```
! Generate the chart;
@CHART(
  'CX CY',      ! Data series;
  'CURVE',      ! Use CURVE chart type;
  'Profit vs. Labor Available', ! Chart title;
  'Y = Profit', ! Label for Y axis;
  'Labor Available' ! Label for X axis;
);
ENDCALC
```

The following graph results.



All the different chart types can be listed by clicking on:

Edit -> Paste Function -> Charting.

For more details on programming in LINGO, see the online documentation or the LINGO manual.

5.9 Problems

1. You wish to represent the status of an academic institution during a specific teaching term. The major features to be represented are that instructors teach courses and students are registered for courses. You want to keep track of who is teaching which course, who is registered for each course, and which courses a given student is taking. What sets would you recommend if each course is taught by exactly one instructor?
2. Suppose we take into account the additional complication of team teaching. That is, two or more instructors teach some courses. How would you modify your answer to the previous question?

6

Product Mix Problems

6.1 Introduction

Product mix problems are conceptually the easiest constrained optimization problems to comprehend. The Astro/Cosmo problem considered earlier is an example. Although product mix problems are seldom encountered in their simple textbook form in practice, they very frequently constitute important components of larger problems such as multiperiod planning models.

The features of a product mix problem are that there is a collection of products competing for a finite set of resources. If there are m resources and n products, then the so-called “technology” is characterized by a table with m rows and n columns of technologic coefficients. The coefficient in row i , column j , is the number of units of resource i used by each unit of product j . The numbers in a row of the table are simply the coefficients of a constraint in the LP. In simple product mix problems, these coefficients are nonnegative. Additionally, associated with each product is a profit contribution per unit and associated with each resource is an availability. The objective is to find how much to produce of each product (i.e., the mix) to maximize profits subject to not using more of each resource than is available.

The following product mix example will illustrate not only product mix LP formulations, but also: 1) representation of nonlinear profit functions and 2) the fact that most problems have alternative correct formulations. Two people may develop different formulations of the same problem, but both may be correct.

6.2 Example

A certain plant can manufacture five different products in any combination. Each product requires time on each of three machines in the following manner (figures in minutes/unit):

Product	Machine		
	1	2	3
A	12	8	5
B	7	9	10
C	8	4	7
D	10	0	3
E	7	11	2

Each machine is available 128 hours per week.

Products *A*, *B*, and *C* are purely competitive and any amounts made may be sold at respective prices of \$5, \$4, and \$5. The first 20 units of *D* and *E* produced per week can be sold at \$4 each, but all made in excess of 20 can only be sold at \$3 each. Variable labor costs are \$4 per hour for machines 1 and 2, while machine 3 labor costs \$3 per hour. Material costs are \$2 for products *A* and *C*, while products *B*, *D*, and *E* only cost \$1. You wish to maximize profit to the firm.

The principal complication is that the profit contributions of products *D* and *E* are not linear. You may find the following device useful for eliminating this complication. Define two additional products *D*₂ and *E*₂, which sell for \$3 per unit. What upper limits must then be placed on the sale of the original products *D* and *E*? The decision variables and their profit contributions are as follows:

Decision Variables	Definition	Profit Contribution per Unit
<i>A</i>	Number of units of <i>A</i> produced per week	$5 - 2 = \$3$
<i>B</i>	Number of units of <i>B</i> produced per week	$4 - 1 = \$3$
<i>C</i>	Number of units of <i>C</i> produced per week	$5 - 2 = \$3$
<i>D</i>	Number of units of <i>D</i> not in excess of 20 produced/week	\$3
<i>D</i> ₂	Number of units of <i>D</i> produced in excess of 20 per week*	\$2
<i>E</i>	Number of units of <i>E</i> not in excess of 20 produced/week	\$3
<i>E</i> ₂	Number of units of <i>E</i> produced in excess of 20	\$2
<i>M</i> ₁	Hours of machine 1 used per week	-\$4
<i>M</i> ₂	Hours of machine 2 used per week	-\$4
<i>M</i> ₃	Hours of machine 3 used per week	-\$3

*Total production of product *D* is $D + D_2$.

We will not worry about issues of sequencing the various products on each machine. This is reasonable if the due-dates for the products are far enough in the future. Our problem in this case is to:

Maximize Revenues minus costs

Subject to

Minutes used equals minutes run on each machine,

At most 20 units each can be produced of products D and E ,

Each machine can be run at most 128 hours.

More precisely, the formulation in LINGO is:

```

! Maximize revenue minus costs;
MAX = 3 * A + 3 * B + 3 * C + 3 * D + 2 * D2 + 3 * E
      + 2 * E2 - 4 * M1 - 4 * M2 - 3 * M3;
! Machine time used = machine time made available;
12*A + 7*B + 8*C + 10*D + 10*D2 + 7*E + 7*E2 - 60*M1 = 0;
8*A + 9*B + 4*C + 11*D + 11*D2 + 60*M2 = 0;
5*A + 10*B + 7*C + 3*D + 3*D2 + 2*E + 2*E2 - 60*M3=0;
D <= 20; ! Max sellable at high price;
E <= 20;
!Machine availability;
M1 <= 128;
M2 <= 128;
M3 <= 128;
END

```

The first three constraints have the units of “minutes” and specify the hours of machine time as a function of the number of units produced. The next two constraints place upper limits on the number of high profit units of D and E that may be sold. The final three constraints put upper limits on the amount of machine time that may be used and have the units of “hours”.

Constraint 2 can be first written as:

$$\frac{12A + 7B + 8C + 10D + 10D_2 + 7E + 7E_2}{60} = M_1$$

Multiplying by 60 and bringing M_1 to the left gives the second constraint. The solution is:

Optimal solution found at step:	4	
Objective value:	1777.625	
Variable	Value	Reduced Cost
A	0.0000000	1.358334
B	0.0000000	0.1854168
C	942.5000	0.0000000
D	0.0000000	0.1291668
D2	0.0000000	1.129167
E	20.00000	0.0000000
E2	0.0000000	0.9187501
M1	128.0000	0.0000000
M2	66.50000	0.0000000
M3	110.6250	0.0000000

Row	Slack or Surplus	Dual Price
1	1777.625	1.000000
2	0.0000000	0.2979167
3	0.0000000	0.6666667E-01
4	0.0000000	0.5000000E-01
5	20.00000	0.0000000
6	0.0000000	0.8125000E-01
7	0.0000000	13.87500
8	61.50000	0.0000000
9	17.37500	0.0000000

The form of the solution is quite simple to state: make as many of E as possible (20). After that, make as much of product C as possible until we run out of capacity on machine 1.

This problem is a good example of one for which it is very easy to develop alternative formulations of the same problem. These alternative formulations are all correct, but may have more or less constraints and variables. For example, the constraint:

$$8A + 9B + 4C + 11E + 11E_2 - 60M_2 = 0$$

can be rewritten as:

$$M_2 = (8A + 9B + 4C + 11E + 11E_2)/60.$$

The expression on the right-hand side can be substituted for M_2 wherever M_2 appears in the formulation. Because the expression on the right-hand side will always be nonnegative, the nonnegativity constraint on M_2 will automatically be satisfied. Thus, M_2 and the above constraint can be eliminated from the problem if we are willing to do a bit of arithmetic. When similar arguments are applied to M_1 and M_3 and the implied divisions are performed, one obtains the formulation:

```

MAX = 1.416667*A + 1.433333*B + 1.85*C + 2.183334*D + 1.183333*D2 +
1.7*E + .7*E2;
! Machine time used = machine time made available;
12*A + 7*B + 8*C + 10*D + 10*D2 + 7*E + 7*E2 <= 7680;
8*A + 9*B + 4*C + 11*E + 11*E2 <= 7680;
5*A + 10*B + 7*C + 3*D + 3*D2 + 2*E + 2*E2 <= 7680;
! Product limits;
D < 20;
E < 20;

```

This looks more like a standard product mix formulation. All the constraints are capacity constraints of some sort. Notice the solution to this formulation is really the same as the previous formulation:

Optimal solution found at step:		6
Objective value:		1777.625
Variable	Value	Reduced Cost
A	0.0000000	1.358333
B	0.0000000	0.1854170
C	942.5000	0.0000000
D	0.0000000	0.1291660
D2	0.0000000	1.129167
E	20.00000	0.0000000
E2	0.0000000	0.9187500
Row	Slack or Surplus	Dual Price
1	1777.625	1.000000
2	0.0000000	0.2312500
3	3690.000	0.0000000
4	1042.500	0.0000000
5	20.00000	0.0000000
6	0.0000000	0.8125000E-01

The lazy formulator might give the first formulation, whereas the second formulation might be given by the person who enjoys doing arithmetic.

6.3 Process Selection Product Mix Problems

A not uncommon feature of product mix models is two or more distinct variables in the LP formulation may actually correspond to alternate methods for producing the same product. In this case, the LP is being used not only to discover how much should be produced of a product, but also to select the best process for producing each product.

A second feature that usually appears with product mix problems is a requirement that a certain amount of a product be produced. This condition takes the problem out of the realm of simple product mix. Nevertheless, let us consider a problem with the above two features.

The American Metal Fabricating Company (AMFC) produces various products from steel bars. One of the initial steps is a shaping operation performed by rolling machines. There are three machines available for this purpose, the B_3 , B_4 , and B_5 . The following table gives their features:

Machine	Speed in Feet per Minute	Allowable Raw Material Thickness in Inches	Available Hours per Week	Labor Cost Per Hour Operating
B_3	150	3/16 to 3/8	35	\$10
B_4	100	5/16 to 1/2	35	\$15
B_5	75	3/8 to 3/4	35	\$17

This kind of combination of capabilities is not uncommon. That is, machines that process larger material operate at slower speed.

This week, three products must be produced. AMFC must produce at least 218,000 feet of $\frac{1}{4}$ " material, 114,000 feet of $\frac{3}{8}$ " material, and 111,000 feet of $\frac{1}{2}$ " material. The profit contributions per foot excluding labor for these three products are 0.017, 0.019, and 0.02. These prices apply to all production (e.g., any in excess of the required production). The shipping department has a capacity limit of 600,000 feet per week, regardless of the thickness.

What are the decision variables and constraints for this problem? The decision variables require some thought. There is only one way of producing $\frac{1}{4}$ " material, three ways of producing $\frac{3}{8}$ ", and two ways of producing $\frac{1}{2}$ ". Thus, you will want to have at least the following decision variables. For numerical convenience, we measure length in thousands of feet:

$$B_{34} = 1,000\text{'s of feet of } \frac{1}{4}\text{" produced on } B_3,$$

$$B_{38} = 1,000\text{'s of feet of } \frac{3}{8}\text{" produced on } B_3,$$

$$B_{48} = 1,000\text{'s of feet of } \frac{3}{8}\text{" produced on } B_4,$$

$$B_{58} = 1,000\text{'s of feet of } \frac{3}{8}\text{" produced on } B_5,$$

$$B_{42} = 1,000\text{'s of feet of } \frac{1}{2}\text{" produced on } B_4,$$

$$B_{52} = 1,000\text{'s of feet of } \frac{1}{2}\text{" produced on } B_5.$$

For the objective function, we must have the profit contribution including labor costs. When this is done, we obtain:

Variable	Profit Contribution per Foot
B_{34}	0.01589
B_{38}	0.01789
B_{48}	0.01650
B_{58}	0.01522
B_{42}	0.01750
B_{52}	0.01622

Clearly, there will be four constraints corresponding to AMFC's three scarce machine resources and its shipping department capacity. There should be three more constraints due to the production requirements in the three products. For the machine capacity constraints, we want the number of hours required for 1,000 feet processed. For machine B_3 , this figure is $1,000/(60 \text{ min./hr.}) \times (150 \text{ ft./min.}) = 0.111111$ hours per 1,000 ft. Similar figures for B_4 and B_5 are 0.16667 hours per 1,000 ft. and 0.22222 hours per 1,000 feet.

The formulation can now be written:

$$\begin{aligned}
 & \text{Maximize} = 15.89B_{34} + 17.89B_{38} + 16.5B_{48} + 15.22B_{58} + 17.5B_{42} + 16.22B_{52} \\
 & \text{subject to} \\
 & 0.11111B_{34} + 0.11111B_{38} \leq 35 \quad \text{Machine} \\
 & 0.16667B_{48} + 0.16667B_{42} \leq 35 \quad \text{capacities} \\
 & 0.22222B_{58} + 0.22222B_{52} \leq 35 \quad \text{in hours} \\
 & B_{34} + B_{38} + B_{48} + B_{58} + B_{42} + B_{52} \leq 600 \quad \text{Shipping capacity in 1,000's of feet} \\
 & B_{34} \geq 218 \quad \text{Production} \\
 & B_{38} + B_{48} + B_{58} \geq 114 \quad \text{requirements} \\
 & B_{42} + B_{52} \geq 111 \quad \text{in 1,000's of feet}
 \end{aligned}$$

Without the last three constraints, the problem is a simple product mix problem.

It is a worthwhile exercise to attempt to deduce the optimal solution just from cost arguments. The $\frac{1}{4}$ " product can be produced on only machine B_3 , so we know B_{34} is at least 218. The $\frac{3}{8}$ " product is more profitable than the $\frac{1}{4}$ " on machine B_3 . Therefore, we can conclude that $B_{34} = 218$ and B_{38} will take up the slack. The $\frac{1}{2}$ " and the $\frac{3}{8}$ " product can be produced on either B_4 or B_5 . In either case, the $\frac{1}{2}$ " is more profitable per foot, so we know B_{48} and B_{58} will be no greater than absolutely necessary. The question is: What is "absolutely necessary"? The $\frac{3}{8}$ " is more profitably run on B_3 than on B_4 or B_5 . Therefore, it follows that we will satisfy the $\frac{3}{8}$ " demand from B_3 and, if sufficient, the remainder from B_4 and then from B_5 . Specifically, we proceed as follows:

$$\text{Set } B_{34} = 218.$$

This leaves a slack of $35 - 218 \times 0.11111 = 10.78$ hours on B_3 . This is sufficient to produce 97,000 feet of $\frac{3}{8}$ ", so we conclude that:

$$B_{38} = 97.$$

The remainder of the $\frac{3}{8}$ " demand must be made up from either machine B_4 or B_5 . It would appear that it should be done on machine B_4 because the profit contribution for $\frac{3}{8}$ " is higher on B_4 than B_5 . Note, however, that $\frac{1}{2}$ " is also more profitable on B_4 than B_5 by exactly the same amount. Thus, we are indifferent. Let us arbitrarily use machine B_4 to fill the rest of $\frac{3}{8}$ " demand. Thus:

$$B_{48} = 17.$$

Now, any remaining capacity will be used to produce $\frac{1}{2}$ " product. There are $35 - 17 \times 0.16667 = 32.16667$ hours of capacity on B_4 . At this point, we should worry about shipping capacity. We still have capacity for $600 - 218 - 97 - 17 = 268$ in 1,000's of feet. B_{42} is more profitable than B_{52} , so we will make it as large as possible. Namely, $32.16667 / 0.16667 = 193$, so:

$$B_{42} = 193.$$

The remaining shipping capacity is $268 - 193 = 75$, so:

$$B_{52} = 75.$$

Any LP is in theory solvable by similar manual economic arguments, but the calculations could be very tedious and prone to errors of both arithmetic and logic. If we take the lazy route and solve it with LINGO, we get the same solution as our manual one:

Optimal solution found at step: 2		
Objective value:		10073.85
Variable	Value	Reduced Cost
B34	218.00000	0.000000
B38	97.00315	0.000000
B48	16.99685	0.000000
B58	0.00000	0.000000
B42	192.99900	0.000000
B52	75.00105	0.000000
Row	Slack or Surplus	Dual Price
1	10073.85	1.000000
2	0.000000	24.030240
3	0.000000	7.679846
4	18.333270	0.000000
5	0.000000	16.220000
6	0.000000	-3.000000
7	0.000000	-1.000000
8	157.000000	0.000000

Ranges in which the basis is unchanged:

Variable	Objective Coefficient Ranges		
	Current Coefficient	Allowable Increase	Allowable Decrease
B34	15.89000	3.000000	INFINITY
B38	17.89000	INFINITY	2.670000
B48	16.50000	1.000000	0.0
B58	15.22000	0.000000	INFINITY
B42	17.50000	0.0	1.000000
B52	16.22000	1.280000	0.0
Row	Right-hand Side Ranges		
	Current RHS	Allowable Increase	Allowable Decrease
2	35.00000	1.888520	9.166634
3	35.00000	12.50043	13.75036
4	35.00000	INFINITY	18.33327
5	600.0000	82.50053	75.00105
6	218.0000	97.00315	16.99685
7	114.0000	157.0000	16.99685
8	111.0000	157.0000	INFINITY

Notice $B58$ is zero, but its reduced cost is also zero. This means $B58$ could be increased (and $B48$ decreased) without affecting profits. This is consistent with our earlier statement that we were indifferent between using $B48$ and $B58$ to satisfy the $\frac{3}{8}$ " demand.

Below is a sets version of the problem:

```

!This is a sets version of the previous example;
MODEL:
SETS:
  MACHINE / B3, B4, B5 / : HPERWK, TIME;
  !This is the coefficient for the time per day constraint;
  THICKNESS / FOURTH, EIGHT, HALF / : NEED;
  !This is the amount of each thickness needed
  to be produced;
  METHOD ( MACHINE, THICKNESS ) : VOLUME, PROFIT, POSSIBLE;
  !VOLUME is the variable, PROFIT the objective coefficients, and
  POSSIBLE is a Boolean representing whether it is possible to produce
  the given thickness;
ENDSETS
DATA:
  ! Hours/week available on each machine;
  HPERWK =   35,   35,   35;
  ! Hours per 1000 feet for each machine;
  TIME = .11111 .16667 .22222;
  ! Amount needed of each product;
  NEED = 218    114    111;
  ! Profit by product and machine;
  PROFIT = 15.89, 17.89,  0,
            0,      16.5,   17.5,
            0,      15.22, 16.22;
  ! Which products can be made on which machine;
  POSSIBLE =  1,      1,      0,
              0,      1,      1,
              0,      1,      1;
  ! Shipping capacity per day;
  SHPERDAY = 600;
ENDDATA
-----
!Objective function;
MAX = @SUM( METHOD(I,J): VOLUME(I,J) * PROFIT(I,J));
@SUM( METHOD( K, L): VOLUME( K, L)) <= SHPERDAY;
!This is the max amount that can be made each day;
@FOR( MACHINE( N):
  ! Maximum time each machine can be used/week.:
  @SUM( THICKNESS( M):
    POSSIBLE(N,M) * VOLUME(N,M) * TIME(N) )<=HPERWK(N));
  @FOR( THICKNESS( Q) :
  !Must meet demand for each thickness;
  @SUM( MACHINE(P): POSSIBLE(P,Q)*VOLUME(P,Q) )>=NEED(Q));
END

```

6.4 Problems

- Consider a manufacturer that produces two products, Widgets and Frisbees. Each product is made from the two raw materials, polyester and polypropylene. The following table gives the amounts required of each of the two products:

Widgets	Frisbees	Raw Material
3	5	Polyester
6	2	Polypropylene

Because of import quotas, the company is able to obtain only 12 units and 10 units of polyester and polypropylene, respectively, this month. The company is interested in planning its production for the next month. For this purpose, it is important to know the profit contribution of each product. These contributions have been found to be \$3 and \$4 for Widgets and Frisbees, respectively. What should be the amounts of Widgets and Frisbees produced next month?

- The Otto Maddick Machine Tool Company produces two products, muffler bearings and torque amplifiers. One muffler bearing requires $\frac{1}{8}$ hour of assembly labor, 0.25 hours in the stamping department, and 9 square feet of sheet steel. Each torque amplifier requires $\frac{1}{3}$ hour in both assembly and stamping and uses 6 square feet of sheet steel. Current weekly capacities in the two departments are 400 hours of assembly labor and 350 hours of stamping capacity. Sheet steel costs 15 cents per square foot. Muffler bearings can be sold for \$8 each. Torque amplifiers can be sold for \$7 each. Unused capacity in either department cannot be laid off or otherwise fruitfully used.

- a) Formulate the LP useful in maximizing the weekly profit contribution.
- b) It has just been discovered that two important considerations were not included.
 - i. Up to 100 hours of overtime assembly labor can be scheduled at a cost of \$5 per hour.
 - ii. The sheet metal supplier only charges 12 cents per square foot for weekly usage in excess of 5000 square feet.

Which of the above considerations could easily be incorporated in the LP model and how? If one or both cannot be easily incorporated, indicate how you might nevertheless solve the problem.

- Review the solution to the 5-product, 3-machine product mix problem introduced at the beginning of the chapter.
 - a) What is the marginal value of an additional hour of capacity on each of the machines?
 - b) The current selling price of product *A* is \$5. What would the price have to be before we would produce any *A*?
 - c) It would be profitable to sell more of product *E* at \$4 if you could, but it is not profitable to sell *E* at \$3 per unit even though you can. What is the breakeven price at which you would be indifferent about selling any more *E*?
 - d) It is possible to gain additional capacity by renting by the hour automatic versions of each of the three machines. That is, they require no labor. What is the maximum hourly rate you would be willing to pay to rent each of the three types of automatic machines?

4. The Aviston Electronics Company manufactures motors for toys and small appliances. The marketing department is predicting sales of 6,100 units of the Dynamonster motor in the next quarter. This is a new high and meeting this demand will test Aviston's production capacities. A Dynamonster is assembled from three components: a shaft, base, and cage. It is clear that some of these components will have to be purchased from outside suppliers because of limited in-house capacity. The variable in-house production cost per unit is compared with the outside purchase cost in the following table.

Component	Outside Cost	Inside Cost
Shaft	1.21	0.81
Base	2.50	2.30
Cage	1.95	1.45

Aviston's plant consists of three departments. The time requirements in hours of each component in each department if manufactured in-house are summarized in the following table. The hours available for Dynamonster production are listed in the last row.

Component	Cutting Department	Shaping Department	Fabrication Department
Shaft	0.04	0.06	0.04
Base	0.08	0.02	0.05
Cage	0.07	0.09	0.06
Capacity	820	820	820

- a) What are the decision variables?
 - b) Formulate the appropriate LP.
 - c) How many units of each component should be purchased outside?
5. Buster Sod's younger brother, Marky Dee, operates three ranches in Texas. The acreage and irrigation water available for the three farms are shown below:

Farm	Acreage	Water Available (acre feet)
1	400	1500
2	600	2000
3	300	900

Three crops can be grown. However, the maximum acreage that can be grown of each crop is limited by the amount of appropriate harvesting equipment available. The three crops are described below:

Crop	Total Harvesting Capacity (in acres)	Water Requirements (in acre-feet/acre)	Expected Profit (in \$/acre)
Milo	700	6	400
Cotton	800	4	300
Wheat	300	2	100

Any combination of crops may be grown on a farm.

- a) What are the decision variables?
 - b) Formulate the LP.
6. Review the formulation and solution of the American Metal Fabricating process selection/product mix problem in this chapter. Based on the solution report:
- a) What is the value of an additional hour of capacity on the B_4 machine?
 - b) What is the value of an additional 2 hours of capacity on the B_3 machine?
 - c) By how much would one have to raise the profit contribution/1,000 ft. of $\frac{1}{4}$ " material before it would be worth producing more of it?
 - d) If the speed of machine B_5 could be doubled without changing the labor cost, what would it be worth per week? (Note labor on B_5 is \$17/hour.)
7. A coupon recently appeared in an advertisement in the weekend edition of a newspaper. The coupon provided \$1 off the price of any size jar of Ocean Spray cranberry juice. The cost of the weekend paper was more than \$1.

Upon checking at a local store, we found two sizes available as follows:

Size in oz.	Price	Price/oz. w/o Coupon	Price/oz. with Coupon
32	2.09	.0653125	.0340625
48	2.89	.0602083	.039375

What questions, if any, should we ask in deciding which size to purchase? What should be our overall objective in analyzing a purchasing decision such as this?

Covering, Staffing & Cutting Stock Models

7.1 Introduction

Covering problems tend to arise in service industries. The crucial feature is that there is a set of requirements to be covered. We have available to us various activities, each of which helps cover some, but not all, the requirements. The qualitative form of a covering problem is:

Choose a minimum cost set of activities

Subject to

The chosen activities cover all of our requirements.

Some examples of activities and requirement types for various problems are listed below:

Problem	Requirements	Activities
Staff scheduling	Number of people required on duty each period of the day or week.	Work or shift patterns. Each pattern covers some, but not all, periods.
Routing	Each customer must be visited.	Various feasible trips, each of which covers some, but not all, customers.
Cutting of bulk raw material stock (e.g., paper, wood, steel, textiles)	Units required of each finished good size.	Cutting patterns for cutting raw material into various finished good sizes. Each pattern produces some, but not every, finished good.

In the next sections, we look at several of these problems in more detail.

7.1.1 Staffing Problems

One part of the management of most service facilities is the scheduling or staffing of personnel. That is, deciding how many people to use on what shifts. This problem exists in staffing the information operators department of a telephone company, a toll plaza, a large hospital, and, in general, any facility that must provide service to the public.

The solution process consists of at least three parts: (1) Develop good forecasts of the number of personnel required during each hour of the day or each day of the week during the scheduling period. (2) Identify the possible shift patterns, which can be worked based on the personnel available and work agreements and regulations. A particular shift pattern might be to work Tuesday through Saturday and then be off two days. (3) Determine how many people should work each shift pattern, so costs are minimized and the total number of people on duty during each time period satisfies the requirements determined in (1). All three of these steps are difficult. LP can help in solving step 3.

One of the first published accounts of using optimization for staff scheduling was by Edie (1954). He developed a method for staffing tollbooths for the New York Port Authority. Though old, Edie's discussion is still very pertinent and thorough. His thoroughness is illustrated by his summary (p. 138): "A trial was conducted at the Lincoln Tunnel...Each toll collector was given a slip showing his booth assignments and relief periods and instructed to follow the schedule strictly...At no times did excessive backups occur...The movement of collectors and the opening and closing of booths took place without the attention of the toll sergeant. At times, the number of booths were slightly excessive, but not to the extent previously... Needless to say, there is a good deal of satisfaction..."

7.1.2 Example: Northeast Tollway Staffing Problems

The Northeast Tollway out of Chicago has a toll plaza with the following staffing demands during each 24-hour period:

Hours	Collectors Needed
12 A.M. to 6 A.M.	2
6 A.M. to 10 A.M.	8
10 A.M. to Noon	4
Noon to 4 P.M.	3
4 P.M. to 6 P.M.	6
6 P.M. to 10 P.M.	5
10 P.M. to 12 Midnight	3

Each collector works four hours, is off one hour, and then works another four hours. A collector can be started at any hour. Assuming the objective is to minimize the number of collectors hired, how many collectors should start work each hour?

Formulation and Solution

Define the decision variables:

x_1 = number of collectors to start work at 12 midnight,

x_2 = number of collectors to start work at 1 A.M.,

.

.

x_{24} = number of collectors to start work at 11 P.M.

There will be one constraint for each hour of the day, which states the number of collectors on at that hour be the number required for that hour. The objective will be to minimize the number of collectors hired for the 24-hour period. More formally:

Minimize $x_1 + x_2 + x_3 + \dots + x_{24}$

subject to

$$x_1 + x_{24} + x_{23} + x_{22} + x_{20} + x_{19} + x_{18} + x_{17} \geq 2 \quad (12 \text{ midnight to } 1 \text{ A.M.})$$

$$x_2 + x_1 + x_{24} + x_{23} + x_{21} + x_{20} + x_{19} + x_{18} \geq 2 \quad (1 \text{ A.M. to } 2 \text{ A.M.})$$

$$\dots$$

$$x_7 + x_6 + x_5 + x_4 + x_2 + x_1 + x_{24} + x_{23} \geq 8 \quad (6 \text{ A.M. to } 7 \text{ A.M.})$$

$$\dots$$

$$x_{24} + x_{23} + x_{22} + x_{21} + x_{19} + x_{18} + x_{17} + x_{16} \geq 3 \quad (11 \text{ P.M. to } 12 \text{ midnight})$$

It may help to see the effect of the one hour off in the middle of the shift by looking at the "PICTURE" of equation coefficients:

Constraint Row	X₁	X₂	X₃	X₄	X₅	X₆	X₇	X₈	X₉	...	X₁₇	X₁₈	X₁₉	X₂₀	X₂₁	X₂₂	X₂₃	X₂₄	RHS
12 A.M. to 1 A.M.	1										1	1	1	1	1	1	1	1	≥2
1 A.M. to 2 A.M.	1	1									1	1	1	1	1	1	1	1	≥2
2 A.M. to 3 A.M.	1	1	1								1	1	1	1	1	1	1	1	≥2
3 A.M. to 4 A.M.	1	1	1	1							1	1	1	1	1	1	1	1	≥2
4 A.M. to 5 A.M.	1	1	1	1	1						1	1	1	1	1	1	1	1	≥2
5 A.M. to 6 A.M.	1		1	1	1	1					1	1	1	1	1	1	1	1	≥2
6 A.M. to 7 A.M.	1	1		1	1	1	1				1	1	1	1	1	1	1	1	≥8
7 A.M. to 8 A.M.	1	1	1		1	1	1	1			1	1	1	1	1	1	1	1	≥8
8 A.M. to 9 A.M.	1	1	1	1		1	1	1	1										≥8
9 A.M. to 10 A.M.	1	1	1	1	1		1	1	1										≥8
10 A.M. to 11 A.M.	1	1	1	1	1		1	1	1										≥4
11 A.M. to 12 P.M.		1	1	1	1		1	1	1										≥4
12 P.M. to 1 P.M.			1	1	1	1		1	1										≥3
1 P.M. to 2 P.M.				1	1	1	1		1										≥3
2 P.M. to 3 P.M.					1	1	1		1										≥3
	<i>etc.</i>												<i>etc.</i>						

Sets Based Formulation

A “sets” based formulation for this problem in LINGO is quite compact. There are two sets, one for the 24-hour day and the other for the nine-hour shift. Note the use of the @WRAP function to modulate the index for the X variable:

```

MODEL: ! 24 hour shift scheduling;
SETS: !Each shift is 4 hours on, 1 hour off, 4 hours on;
      HOUR/1..24/: X, NEED;
ENDSETS
DATA:
  NEED=2 2 2 2 2 2 8 8 8 8 4 4 3 3 3 3 6 6 5 5 5 5 3 3;
ENDDATA
MIN = @SUM( HOUR(I): X(I));
@FOR( HOUR( I): ! People on duty in hour I are those who
      started 9 or less hours earlier, but not 5;
  @SUM(HOUR(J) | (J#LE#9) #AND# (J#NE#5): X(@WRAP((I-J+1),24)))>= NEED(I));
END

```

When solved as an LP, we get an objective value of 15.75 with the following variables nonzero:

$$\begin{array}{llll}
 x_2 = 5 & x_5 = 0.75 & x_{11} = 1 & x_{16} = 1 \\
 x_3 = 0.75 & x_6 = 0.75 & x_{14} = 1 & x_{17} = 1 \\
 x_4 = 0.75 & x_7 = 0.75 & x_{15} = 2 & x_{18} = 1
 \end{array}$$

The answer is not directly useful because some of the numbers are fractional. To enforce the integrality restriction, use the @GIN function as in the following line:

```
@FOR( HOUR(I): @GIN( X(I)) );
```

When it is solved, we get an objective value of 16 with the following variables nonzero:

$$\begin{array}{llll}
 x_2 = 4 & x_5 = 1 & x_{14} = 1 & x_{17} = 2 \\
 x_3 = 1 & x_6 = 1 & x_{15} = 1 & x_{18} = 1 \\
 x_4 = 1 & x_7 = 1 & x_{16} = 2
 \end{array}$$

One of the biggest current instances of this kind of staffing problem is in telephone call centers. Examples are telephone order takers for catalog retailers and credit checkers at credit card service centers. A significant fraction of the population of Omaha, Nebraska works in telephone call centers. A typical shift pattern at a call center consists of 8 hours of work split by a 15 minute break, a half hour lunch break, and another 15 minute break.

7.1.3 Additional Staff Scheduling Features

In a complete implementation there may be a fourth step, rostering, in addition to the first three steps of forecasting, work pattern identification, and work pattern selection. In rostering, specific individuals by name are assigned to specific work patterns. In some industries, e.g., airlines, individuals(e.g., pilots) are allowed to bid on the work patterns that have been selected.

In some staffing situations, there may be multiple skill requirements that need to be covered, e.g. during the first hour there must be at least 3 Spanish speakers on duty and at least 4 English speakers. Different employees may have different sets of skills, e.g., some may speak only English, some are conversant in both English and Spanish, etc.

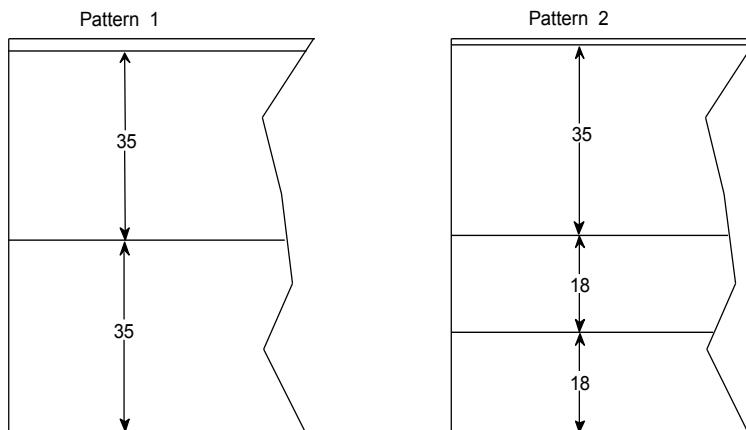
In some situations, e.g., mail processing, demand may be postponable by one or two periods, so that we are allowed to be understaffed, say during a peak period, if the carryover demand can be processed in the next period.

In almost all situations, demand is somewhat random so that staff requirements are somewhat soft. We might say that we need at least ten people on duty during a certain period, however, if we have eleven on duty, the extra person will probably not be standing around idle during the full period. There is a good chance that by chance demand will be higher than the point forecast so that we can use the extra person. The queueing theory methods of chapter 18 are frequently used to provide estimates of the marginal benefit of each unit of overstaffing.

7.2 Cutting Stock and Pattern Selection

In industries such as paper, plastic food wrap, and textiles, products are manufactured in large economically produced sizes at the outset. These sizes are cut into a variety of smaller, more usable sizes as the product nears the consumer. The determination of how to cut the larger sizes into smaller sizes at minimal cost is known as the cutting stock problem. As an example of the so-called one-dimensional cutting stock problem, suppose machine design dictates material is manufactured in 72-inch widths. There are a variety of ways of cutting these smaller widths from the 72-inch width, two of which are shown in Figure 7.1.

Figure 7.1 Example Cutting Patterns



Pattern 1 has 2 inches of edge waste ($72 - 2 \times 35 = 2$), whereas there is only 1 inch of edge waste ($72 - 2 \times 18 - 35 = 1$) with pattern 2. Pattern 2, however, is not very useful unless the number of linear feet of 18-inch material required is about twice the number of linear feet of 35-inch material required. Thus, a compromise must be struck between edge waste and end waste.

The solution of a cutting stock problem can be partitioned into the 3-step procedure discussed earlier: (1) Forecast the needs for the final widths. (2) Construct a large collection of possible patterns for cutting the large manufactured width(s) into the smaller widths. (3) Determine how much of each pattern should be run of each pattern in (2), so the requirements in (1) are satisfied at minimum cost. Optimization can be used in performing step (3).

Many large paper manufacturing firms have LP-based procedures for solving the cutting stock problem. Actual cutting stock problems may involve a variety of cost factors in addition to the edge waste/end waste compromise. The usefulness of the LP-based procedure depends upon the importance of these other factors. The following example illustrates the fundamental features of the cutting stock problem with no complicating cost factors.

7.2.1 Example: Cooldot Cutting Stock Problem

The Cooldot Appliance Company produces a wide range of large household appliances such as refrigerators and stoves. A significant portion of the raw material cost is due to the purchase of sheet steel. Currently, sheet steel is purchased in coils in three different widths: 72 inches, 48 inches, and 36 inches. In the manufacturing process, eight different widths of sheet steel are required: 60, 56, 42, 38, 34, 24, 15, and 10 inches. All uses require the same quality and thickness of steel.

A continuing problem is trim waste. For example, one way of cutting a 72-inch width coil is to slit it into one 38-inch width coil and two 15-inch width coils. There will then be a 4-inch coil of trim waste that must be scrapped.

The prices per linear foot of the three different raw material widths are 15 cents for the 36-inch width, 19 cents for the 48-inch width, and 28 cents for the 72-inch width. Simple arithmetic reveals the costs per inch × foot of the three widths are $15/36 = 0.416667$ cents/(inch × foot), 0.395833 cents/(inch × foot), and 0.388889 cents/(inch × foot) for the 36", 48", and 72" widths, respectively.

The coils may be slit in any feasible solution. The possible cutting patterns for efficiently slitting the three raw material widths are tabulated below.

For example, pattern C_4 corresponds to cutting a 72-inch width coil into one 24-inch width and four 10-inch widths with 8 inches left over as trim waste.

The lengths of the various widths required in this planning period are:

Width	60"	56"	42"	38"	34"	24"	15"	10"
Number of feet required	500	400	300	450	350	100	800	1000

The raw material availabilities this planning period are 1600 ft. of the 72-inch coils and 10,000 ft. each of the 48-inch and 36-inch widths.

How many feet of each pattern should be cut to minimize costs while satisfying the requirements of the various widths? Can you predict beforehand the amount of 36-inch material used?

7.2.2 Formulation and Solution of Cooldot

Let the symbols A_1, A_2, \dots, E_4 appearing in the following table denote the number of feet to cut off the corresponding pattern:

Cutting Patterns for Raw Material

	48-Inch Raw Material								
D_0	0	0	1	0	0	0	0	0	6
D_1	0	0	0	1	0	0	0	1	0
D_2	0	0	0	0	1	0	0	1	4
D_3	0	0	0	0	0	2	0	0	0
D_4	0	0	0	0	0	1	1	0	9
D_5	0	0	0	0	0	1	0	2	4
D_6	0	0	0	0	0	0	3	0	3
D_7	0	0	0	0	0	0	2	1	8
D_8	0	0	0	0	0	0	1	3	3
D_9	0	0	0	0	0	0	0	4	8
	36-Inch Raw Material								
E_0	0	0	0	0	1	0	0	0	2
E_1	0	0	0	0	0	1	0	1	2
E_2	0	0	0	0	0	0	2	0	6
E_3	0	0	0	0	0	0	1	2	1
E_4	0	0	0	0	0	0	0	3	6

For accounting purposes, it is useful to additionally define:

- T_1 = number of feet cut of 72-inch patterns,
- T_2 = number of feet cut of 48-inch patterns,
- T_3 = number of feet cut of 36-inch patterns,
- W_1 = inch \times feet of trim waste from 72-inch patterns,
- W_2 = inch \times feet of trim waste from 48-inch patterns,
- W_3 = inch \times feet of trim waste from 36-inch patterns,
- X_1 = number of excess feet cut of the 60-inch width,
- X_2 = number of excess feet cut of the 56-inch width,
- .
- .
- X_8 = number of excess feet cut of the 10-inch width.

It may not be immediately clear what the objective function should be. One might be tempted to calculate a cost of trim waste per foot for each pattern cut and then minimize the total trim waste cost. For example:

$$\text{MIN} = 0.3888891W_1 + 0.395833W_2 + 0.416667W_3;$$

However, such an objective can easily lead to solutions with very little trim waste, but very high cost. This is possible in particular when the cost per square inch is not the same for all raw material widths. A more reasonable objective is to minimize the total cost. That is:

$$\text{MIN} = 28 * T_1 + 19 * T_2 + 15 * T_3;$$

Incorporating this objective into the model, we have:

```

MODEL:
SETS:
! Each raw material has a Raw material width, Total used,
! Waste total, Cost per unit, Waste cost, and Supply available;
RM: RWDTH,T, W, C, WCOST, S;
! Each Finished good has a Width, units Required. eXtra produced;
FG: FWDTH, REQ, X;
PATTERN: USERM, WASTE, AMT;
PXF( PATTERN, FG): NUM;
ENDSETS
DATA:
! The raw material widths;
RM =      R72      R48      R36;
RWDTH=    72       48       36;
C =       .28      .19      .15;
WCOST=   .00388889 .00395833 .00416667;
S =       1600     10000    10000;
! The finished good widths;
FG = F60 F56 F42 F38 F34 F24 F15 F10;
FWDTH=  60   56   42   38   34   24   15   10;
REQ= 500 400 300 450 350 100 800 1000;
! Index of R.M. that each pattern uses;
USERM = 1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1
        2 2 2 2 2 2 2 2 2 2
        3 3 3 3 3 3;
! How many of each F.G. are in each R.M. pattern;
NUM=   1 0 0 0 0 0 0 1
      0 1 0 0 0 0 1 0
      0 1 0 0 0 0 0 1
      0 0 1 0 0 1 0 0
      0 0 1 0 0 0 2 0
      0 0 1 0 0 0 1 1
      0 0 1 0 0 0 0 3
      0 0 0 1 1 0 0 0
      0 0 0 1 0 1 0 1
      0 0 0 1 0 0 2 0
      0 0 0 1 0 0 1 1
      0 0 0 1 0 0 0 3
      0 0 0 0 2 0 0 0
      0 0 0 0 1 1 0 1
      0 0 0 0 1 0 2 0
      0 0 0 0 1 0 1 2
      0 0 0 0 1 0 0 3
      0 0 0 0 0 3 0 0
      0 0 0 0 0 2 1 0
      0 0 0 0 0 2 0 2
      0 0 0 0 0 1 3 0
      0 0 0 0 0 1 2 1
      0 0 0 0 0 1 1 3
      0 0 0 0 0 1 0 4

```

```

0 0 0 0 0 0 4 1
0 0 0 0 0 0 3 2
0 0 0 0 0 0 2 4
0 0 0 0 0 0 1 5
0 0 0 0 0 0 0 7
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 0 2 0 0
0 0 0 0 0 1 1 0
0 0 0 0 0 1 0 2
0 0 0 0 0 0 3 0
0 0 0 0 0 0 2 1
0 0 0 0 0 0 1 3
0 0 0 0 0 0 0 4
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 1
0 0 0 0 0 0 2 0
0 0 0 0 0 0 1 2
0 0 0 0 0 0 0 3;
ENDDATA
! Minimize cost of raw material used;
MIN = TCOST;
TCOST = @SUM(RM(I): C(I)*T(I) );
! Compute total cost of waste;
TOTWASTE = @SUM( RM(I): WCOST(I)*W(I) );
@FOR( RM( I):
      T( I) = @SUM( PATTERN( K) | USERM(K) #EQ# I: AMT( K));
! Raw material supply constraints;
      T(I) <= S(I);
    );
! Must produce at least amount required of each F.G. ;
@FOR( FG(J):
      @SUM(PATTERN(K): NUM(K,J)*AMT(K)) = REQ(J) + X(J);
    );
! Turn this on to get integer solutions;
!@FOR( PATTERN(K): @GIN(AMT(K)));
! Waste related computations;
! Compute waste associated with each pattern;
@FOR( PATTERN(K):
      WASTE(K) = RWDTH(USERM(K)) - @SUM(FG(J): FWDTH(J)*NUM(K,J));
    );
! Waste for each R.M. in this solution;
@FOR( RM( I):
      W(I) = @SUM( PATTERN( K) | USERM(K) #EQ# I: WASTE(K)*AMT( K));
    );
END

```

If you minimize cost of waste, then you will get a different solution than if you minimize total cost of raw materials. Two different solutions obtained under the two different objectives are compared in the following table:

Cutting Stock Solutions		
Nonzero Patterns	Trim Waste Minimizing Solution Feet to Cut	Total Cost Minimizing Solution Feet to Cut
A_1	500	500
A_2	400	400
A_5	200	171.4286
A_7	100	128.5714
A_8	350	350
A_9	50	3.571429
B_8	0	32.14286
C_9	0	14.28571
D_1	150	96.42857
D_3	25	0
Trim Waste Cost	\$5.44	\$5.55
Total Cost	\$2348.00	\$466.32
X_4	100.000	0
X_6	19650	0
T_1	1600	1600
T_2	10000	96.429
T_3	0	0

The key difference in the solutions is the “Min trim waste” solution uses more of the 48” width raw material, patterns D_1 and D_3 , and cuts in a way so the edge waste is minimized. The “Min trim waste” solution produces more of the 38” width, 550 units, than is needed, 450 units, because the objective function does not count this as waste. The “Min trim waste” formulation has a number of alternate optimal solutions, some having a raw material cost less than \$2348. A key observation from this example is that you should always remember your overall objective, e.g., minimize total cost or maximize total profit, and not get distracted by optimizing secondary criteria.

Both solutions involve fractional answers. By turning on the @GIN declaration you can get an integer answer. The cost of the “cost minimizing” solution increases to \$466.34.

7.2.3 Generalizations of the Cutting Stock Problem

In large cutting stock problems, it may be unrealistic to generate all possible patterns. There is an efficient method for generating only the patterns that have a very high probability of appearing in the optimal solution. It is beyond the scope of this section to discuss this procedure. However, it does become important in large problems. See Chapter 18 for details. Dyckhoff (1981) describes another

formulation that avoids the need to generate patterns. However, that formulation may have a very large number of rows.

Complications

In complex cutting stock problems, the following additional cost considerations may be important:

1. *Fixed cost of setting up a particular pattern.* This cost consists of lost machine time, labor, etc. This motivates solutions with few patterns.
2. *Value of overage or end waste.* For example, there may be some demand next period for the excess cut this period.
3. *Underage cost.* In some industries, you may supply plus or minus, say 5%, of a specified quantity. The cost of producing the least allowable amount is measured in foregone profits.
4. *Machine usage cost.* The cost of operating a machine is usually fairly independent of the material being run. This motivates solutions that cut up wide raw material widths.
5. *Material specific products.* It may be impossible to run two different products in the same pattern if they require different materials (e.g., different thickness, quality, surface finish or type).
6. *Upgrading costs.* It may be possible to reduce setup, edge-waste, and end-waste costs by substituting a higher-grade material than required for a particular demand width.
7. *Order splitting costs.* If a demand width is produced from several patterns, then there will be consolidation costs due to bringing the different lots of the output together for shipment.
8. *Stock width change costs.* A setup involving only a pattern change usually takes less time than one involving both a pattern change and a raw material width change. This motivates solutions that use few raw material widths.
9. *Minimum and maximum allowable edge waste.* For some materials, a very narrow ribbon of edge waste may be very difficult to handle. Therefore, one may wish to restrict attention to patterns that have either zero edge waste or edge waste that exceeds some minimum, such as two centimeters. On the other hand, one may also wish to specify a maximum allowable edge waste. For example, in the paper industry, edge waste may be blown down a recycling chute. Edge waste wider than a certain minimum may be too difficult to blow down this chute.
10. *Due dates and sequencing.* Some of the demands need to be satisfied immediately, whereas others are less urgent. The patterns containing the urgent or high priority products should be run first. If the urgent demands appear in the same patterns as low priority demands, then it is more difficult to satisfy the high priority demands quickly.
11. *Inventory restrictions.* Typically, a customer's order will not be shipped until all the demands for the customer can be shipped. Thus, one is motivated to distribute a given customer's demands over as few patterns as possible. If every customer has product in every pattern, then no customer's order can be shipped until every pattern has been run. Thus, there will be substantial work in process inventory until all patterns have been run.
12. *Limit on 1-set patterns.* In some industries, such as paper, there is no explicit cost associated with setting up a pattern, but there is a limit on the rate at which pattern changes can be made. It may take about 15 minutes to do a pattern change, much of this work being done off-line without shutting down the main machine. The run time to produce one roll set might take 10 minutes. Thus, if too many 1-set patterns are run, the main machine will have to wait for pattern changes to be completed.

13. *Pattern restrictions.* In some applications, there may be a limit on the total number of final product widths that may appear in a pattern, and/or a limit on the number of “small” widths in a pattern. The first restriction would apply, for example, if there were a limited number of take-up reels for winding the slit goods. The second restriction might occur in the paper industry where rolls of narrow product width have a tendency to fall over, so one does not want to have too many of them to handle in a single pattern. Some demanding customers may request their product be cut from a particular position (e.g., the center) of a pattern, because they feel the quality of the material is higher in that position.
14. *Pattern pairing.* In some plastic wrap manufacturing, the production process, by its nature, produces two widths of raw material simultaneously, an upper output and a lower output. Thus, it is essentially unavoidable that one must run the same number of feet of whatever pattern is being used on the upper output as on the lower output. A similar situation sometimes happens by accident in paper manufacturing. If a defect develops on the “production belt”, a small width of paper in the interior of the width is unusable. Thus, the machine effectively produces two output widths, one to the left of the defect, the other to the right of the defect.
15. *Bundle size and/or minimum purchase quantity.* In some markets you may be forced to buy product in bundles of a given size, e.g., 10 pieces per bundle. Further, you may be forced to make cuts in bundles rather than in individual pieces. Thus, even though you have a demand of 15 units for some finished good, you are forced to cut at least 20 units because the bundle size is 10.
16. *Saw thickness/kerf.* If the material is sawed rather than sheared, each saw cut may remove a small amount of material, sometimes called the kerf. Precise solution of a cutting stock problem should take into account material lost to the kerf.
17. *Max “smalls” per pattern.* There may be a limit on the number of “small” widths in a pattern. This restriction might be encountered in the paper industry where rolls of narrow product width have a tendency to tip over, so one does not want to have too many of them to handle in a single pattern.

Most of the above complications can be incorporated by making modest changes to the pattern generating procedure. The most troublesome complications are high fixed setup costs, order-splitting costs, and stock width change costs. If they are important, then one will usually be forced to use some ad hoc, manual solution procedure. An LP solution may provide some insight into which solutions are likely to be good, but other methods must be used to determine a final workable solution.

7.2.4 Two-Dimensional Cutting Stock Problems

The one-dimensional cutting stock problem is concerned with the cutting of a raw material that is in coils. The basic idea still applies if the raw material comes in sheets and the problem is to cut these sheets into smaller sheets. For example, suppose plywood is supplied in 48- by 96-inch rectangular sheets and the end product demand is for sheets with dimensions in inches of 36×50 , 24×36 , 20×60 , and 18×30 . Once you have enumerated all possible patterns for cutting a 48×96 sheet into combinations of the four smaller sheets, then the problem is exactly as before.

Enumerating all possible two-dimensional patterns may be a daunting task. Two features of practical two-dimensional cutting problems can reduce the size of this task: (a) orientation requirements, and (b) “guillotine” cut requirements. Applications in which (a) is important are in the

cutting of wood and fabric. For reasons of strength or appearance, a demand unit may be limited in how it is positioned on the raw material (Imagine a plaid suit for which the manufacturer randomly oriented the pattern on the raw material). Any good baseball player knows the grain of the bat must face the ball when hitting. Attention must be paid to the grain of the wood if the resulting wood product is to be used for structural or aesthetic purposes. Glass is an example of a raw material for which orientation is not important. A pattern is said to be cuttable with guillotine cuts if each cut must be made by a shear across the full width of the item being cut.

7.3 Crew Scheduling Problems

A major component of the operating cost of an airline is the labor cost of its crews. Managing the aircraft and crews of a large airline is a complex scheduling problem. Paying special attention to these scheduling problems can be rewarding. The yearly cost of a crew member far exceeds the one-time cost of a typical computer, so devoting some computing resources to make more efficient use of crews and airplanes is attractive. One small part of an airline's scheduling problems is discussed below.

Large airlines face a staffing problem known as the crew-scheduling problem. The requirements to be covered are the crew requirements of the flights that the airline is committed to fly during the next scheduling period (e.g., one month). During its working day, a specific crew typically, but not necessarily, flies a number of flights on the same aircraft. The problem is to determine which flights should comprise the day's work of a crew.

The approach taken by many airlines is similar to the approach described for general staffing problems: (1) Identify the demand requirements (i.e., the flights to be covered). (2) Generate a large number of feasible sequences of flights one crew could cover in a work period. (3) Select a minimum cost subset of the collections generated in (2), so the cost is minimized and every flight is contained in exactly one of the selected collections.

Integer programming(IP) can be used for step (3). Until 1985, most large airlines used computerized ad hoc or heuristic procedures for solving (3) because the resulting IP tends to be large and difficult to solve. Marsten, Muller, and Killion (1979), however, described an IP-based solution procedure that was used very successfully by Flying Tiger Airlines. Flying Tiger had a smaller fleet than the big passenger carriers, so the resulting IP could be economically solved and gave markedly lower cost solutions than the ad hoc, heuristic methods. These optimizing methods are now being extended to large airlines.

A drastically simplified version of the crew-scheduling problem is given in the following example. This example has only ten flights to be covered. By contrast, a typical major airline has over 2000 flights per day to be covered.

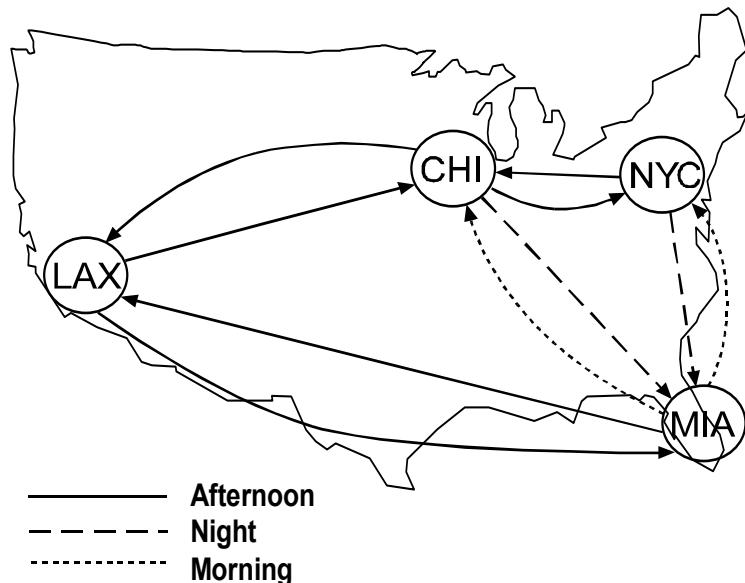
7.3.1 Example: Sayre-Priors Crew Scheduling

The Sayre-Priors Airline and Stormdoor Company is a small diversified company that operates the following set of scheduled flights:

Flights			
Flight Number	Origin	Destination	Time of Day
101	Chicago	Los Angeles	Afternoon
410	New York	Chicago	Afternoon
220	New York	Miami	Night
17	Miami	Chicago	Morning
7	Los Angeles	Chicago	Afternoon
13	Chicago	New York	Night
11	Miami	New York	Morning
19	Chicago	Miami	Night
23	Los Angeles	Miami	Night
3	Miami	Los Angeles	Afternoon

The flight schedule is illustrated graphically in the figure below:

Figure: Flight Schedule



The Flight Operations Staff would like to set up a low-cost crew assignment schedule. The basic problem is to determine the next flight, if any, a crew operates after it completes one flight. A basic concept needed in understanding this problem is that of a tour. The characteristics of a tour are as follows:

- A tour consists of from one to three connecting flights.
- A tour has a cost of \$2,000 if it terminates in its city of origin.
- A tour that requires “deadheading” (i.e., terminates in a city other than the origin city) costs \$3,000.

In airline parlance, a tour is frequently called a “pairing” or a “rotation” because a tour consist of a pair of duty periods, an outbound one, and a return one. The following are examples of acceptable tours:

Tour	Cost
17, 101, 23	\$2,000
220, 17, 101	\$3,000
410, 13	\$2,000

In practice, the calculation of the cost of a tour is substantially more complicated than above. There might be a minimum fixed payment, c_f , to a crewmember simply for being on duty, a guaranteed rate per hour, c_d , while in the airplane, and a guaranteed rate per hour, c_e , for total elapsed time away from home so that the we might have: $\text{pairing_cost}(i) = \max\{ c_f, c_d * \text{flying_time}(i), c_e * \text{elapsed_time}(i) \}$.

7.3.2 Solving the Sayre/Priors Crew Scheduling Problem

The first thing to do for this small problem is to enumerate all feasible tours. We do not consider a collection of flights that involve an intermediate layover a tour. There are 10 one-flight tours, 14 two-flight tours, and either 37 or 41 three-flight tours depending upon whether one distinguishes the origin city on a non-deadheading tour. These tours are indicated in the following table:

List of Tours

One-Flight Tours	Cost	Two-Flight Tours	Cost	Three-Flight Tours	Cost
1. 101	\$3,000	11. 101, 23	\$3,000	25. 101, 23, 17	\$2,000
2. 410	\$3,000	12. 410, 13	\$2,000	26. 101, 23, 11	\$3,000
3. 220	\$3,000	13. 410, 19	\$3,000	27. 410, 19, 17	\$3,000
4. 17	\$3,000	14. 220, 17	\$3,000	28. 410, 19, 11	\$2,000
5. 7	\$3,000	15. 220, 11	\$2,000	29. 220, 17, 101	\$3,000
6. 13	\$3,000	16. 17, 101	\$3,000	30. 220, 11, 410	\$3,000
7. 11	\$3,000	17. 7, 13	\$3,000	25. 17, 101, 23	\$2,000
8. 19	\$3,000	18. 7, 19	\$3,000	31. 7, 19, 17	\$3,000
9. 23	\$3,000	19. 11, 410	\$3,000	32. 7, 19, 11	\$3,000
10. 3	\$3,000	20. 19, 17	\$2,000	33. 11, 410, 13	\$3,000
		21. 19, 11	\$3,000	28. 11, 410, 19	\$2,000
		22. 23, 17	\$3,000	34. 19, 17, 101	\$3,000
		23. 23, 11	\$3,000	28. 19, 11, 410	\$2,000
		24. 3, 23	\$2,000	25. 23, 17, 101	\$2,000
				35. 23, 11, 410	\$3,000
				36. 3, 23, 17	\$3,000
				37. 3, 23, 11	\$3,000

Define the decision variables:

$$T_i = \begin{cases} 1 & \text{if tour } i \text{ is used} \\ 0 & \text{if tour } i \text{ is not used, for } i = 1, 2, \dots, 37. \end{cases}$$

We do not distinguish the city of origin on non-deadheading three-flight tours. The formulation, in words, is:

Minimize the cost of the tours selected;

Subject to,

For each flight leg i :

The number of tours selected must include exactly one that covers flight i .

In LINGO scalar format, a model is:

```

MODEL:
[_1] MIN= 3 * T_1 + 3 * T_2 + 3 * T_3 + 3 * T_4 + 3 * T_5
    + 3 * T_6 + 3 * T_7 + 3 * T_8 + 3 * T_9 + 3 * T_10
    + 3 * T_11 + 2 * T_12 + 3 * T_13 + 3 * T_14 + 2 * T_15
    + 3 * T_16 + 3 * T_17 + 3 * T_18 + 3 * T_19 + 2 * T_20
    + 3 * T_21 + 3 * T_22 + 3 * T_23 + 2 * T_24 + 2 * T_25
    + 3 * T_26 + 3 * T_27 + 2 * T_28 + 3 * T_29 + 3 * T_30
    + 3 * T_31 + 3 * T_32 + 3 * T_33 + 3 * T_34 + 3 * T_35
    + 3 * T_36 + 3 * T_37 ;
[COV_F101] T_1 + T_11 + T_16 + T_25 + T_26 + T_29 + T_34 = 1 ;
[COV_F410] T_2 + T_12 + T_13 + T_19 + T_27 + T_28 + T_30
    + T_33 + T_35 = 1 ;
[COV_F220] T_3 + T_14 + T_15 + T_29 + T_30 = 1 ;
[COV_F17] T_4 + T_14 + T_16 + T_20 + T_22 + T_25 + T_27
    + T_29 + T_31 + T_34 + T_36 = 1 ;
[COV_F7] T_5 + T_17 + T_18 + T_31 + T_32 = 1 ;
[COV_F13] T_6 + T_12 + T_17 + T_26 = 1 ;
[COV_F11] T_7 + T_15 + T_19 + T_21 + T_23 + T_26 + T_28
    + T_30 + T_32 + T_33 + T_35 + T_37 = 1 ;
[COV_F19] T_8 + T_13 + T_18 + T_20 + T_21 + T_27 + T_28
    + T_31 + T_32 + T_34 = 1 ;
[COV_F23] T_9 + T_11 + T_22 + T_23 + T_24 + T_25 + T_35
    + T_36 + T_37 = 1 ;
[COV_F3] T_10 + T_24 + T_36 + T_37 = 1 ;
@BIN( T_1); @BIN( T_2); @BIN( T_3); @BIN( T_4); @BIN( T_5);
@BIN( T_6); @BIN( T_7); @BIN( T_8); @BIN( T_9); @BIN( T_10);
@BIN( T_11); @BIN( T_12); @BIN( T_13); @BIN( T_14); @BIN( T_15);
@BIN( T_16); @BIN( T_17); @BIN( T_18); @BIN( T_19); @BIN( T_20);
@BIN( T_21); @BIN( T_22); @BIN( T_23); @BIN( T_24); @BIN( T_25);
@BIN( T_26); @BIN( T_27); @BIN( T_28); @BIN( T_29); @BIN( T_30);
@BIN( T_31); @BIN( T_32); @BIN( T_33); @BIN( T_34); @BIN( T_35);
@BIN( T_36); @BIN( T_37);

END

```

The tabular “Picture” below of the coefficients may give a better feel for the structure of the problem. The first constraint, for example, forces exactly one of the tours including flight 101, to be chosen:

When solved simply as an LP, the solution is naturally integer, with the tours selected being:

Tour	Flights
T17	7, 13
T24	3, 23
T28	410, 19, 11
T29	220, 17, 101

The cost of this solution is \$10,000.

It can be shown, e.g., by adding the constraint: $T_{17} + T_{24} + T_{28} + T_{29} \leq 3$, that there is one other solution with a cost of \$10,000, namely:

Tour	Flights
T12	410, 13
T24	3, 23
T29	220, 17, 101
T32	7, 11, 19

The kind of IP's that result from the crew scheduling problem with more than 500 constraints have proven somewhat difficult to solve. A general formulation of the Sayre/Priors problem is given in section 7.4.

7.3.3 Additional Practical Details

An additional detail sometimes added to the above formulation in practice is crew-basing constraints. Associated with each tour is a home base. Given the number of pilots living near each home base, one may wish to add a constraint for each home base that puts an upper limit on the number of tours selected for that home base.

A simplification in practice is that a given pilot is typically qualified for only one type of aircraft. Thus, a separate crew-scheduling problem can be solved for each aircraft fleet (e.g., Boeing 747, Airbus 320, etc.). Similarly, cabin attendant crews can be scheduled independently of flight (pilot) crews.

After crew schedules have been selected, there still remains the rostering problem of assigning specific crew schedules to specific pilots. In the U.S., perhaps by union agreement, many of the major airlines allow pilots to select schedules by a bidding process. In some smaller airlines and in non-U.S. airlines schedules may be assigned to specific pilots by a central planning process. The bidding process may make some of the crew members more happy, at least the ones with high seniority, however, the centralized assignment process may be more efficient because it eliminates inefficient gaming of the system.

Uncertainties due to bad weather and equipment breakdowns are an unfortunate fact of airline life. Thus, one wishes to have crew schedules that are robust or insensitive to disruptions. A crew schedule tends to be disruption sensitive if the schedule requires crews to change planes frequently and if the connection times between these changes are short. Suppose that a plane is delayed a half hour by a need to get some equipment repaired. If the crew of this plane is scheduled to change planes at the next stop, then at the next stop the airline may need or wish to delay up to three flights: a) the flight that is scheduled to use this plane, b) the flight which is scheduled to use this flight crew, and c) the flight to which a significant number of passengers on this flight are transferring. In our little example we saw that there were alternate optima. Thus, in addition to minimizing total crew costs,

we might wish to secondarily minimize expected delays by avoiding tours that involve crews changing planes a) such that there is short change-over time, b) different from the plane to which most of the passengers are changing (thus two flights must be delayed), c) at an airport where there are few backup crews available to take over for the delayed crew.

7.4 A Generic Covering/Partitioning/Packing Model

The model given for the crew-scheduling problem was very specific to that particular problem. The following is a fairly general one for any standard, so-called covering, partitioning or packing problem. This model takes the viewpoint of facilities and customers. Each facility or pattern, if opened, serves a specified set of customers or demands. Specialized to our crew-scheduling example, a tour is a facility, and a flight is a customer. The main data to be entered are the "2-tuples" describing which facilities serve which customers. This model allows considerable flexibility in specifying whether customers are over served or under served. If the parameter *BUDU* is set to 0, this means that every customer (or flight) must be served by at least one open facility (or tour). This is sometimes called a set covering problem. Alternatively, if *BUDV* is set to 0, then each flight can appear in at most one selected tour. This is sometimes called a set packing problem. If both *BUDV* and *BUDU* are set to zero, then each flight must appear in exactly one selected tour. This is sometimes called a set partitioning problem. It is called set partition because any feasible solution in fact partitions the set of customers into subsets, one subset for each chosen open facility (or tour).

```

! The set covering/partitioning/packing/ problem (COVERPAK);
SETS:
! Given a set of demands, a set of candidate patterns,
and which demands are covered by each pattern,
which patterns should be used?;
PATTERN: COST, Y; ! The patterns or tours;
DMND: CU, CV, U, V;
! The "which PATTERN serves which demand" 2-tuples;
PXD( PATTERN, DMND);
ENDSETS
DATA:
! Data for a simple crew scheduling problem;
PATTERN = 1..37;
! Cost of each PATTERN;
COST = 3 3 3 3 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 2
          3 3 3 2 2 3 3 2 3 3 3 3 3 3 3 3 3 3;
! Names of the demands;
DMND= F101 F410 F220 F17 F7 F13 F11 F19 F23 F3;
! Cost/unit under at each demand;
CU =    1;
! Cost/unit over at each demand;
CV =    1;

! Max allowed to spend on patterns or facilities facilities;
BUDGET = 9999;
! Max allowed underage at each demand,
      0 makes a covering or partitioning problem;
BUDU = 0;
! Max allowed overage at each demand,
      0 makes it a packing or partitioning problem;

```

```

        BUDV = 0;
! Both = 0 makes it a partitioning problem;
PXD =
1,F101 2,F410 3,F220 4,F17 5,F7 6,F13 7,F11 8,F19 9,F23 10,F3
11,F101 11,F23 12,F410 12,F13 13,F410 13,F19 14,F220 14,F17
15,F220 15,F11 16,F17 16,F101 17,F7 17,F13 18,F7 18,F19
19,F11 19,F410 20,F19 20,F17 21,F19 21,F11 22,F23 22,F17
23,F23 23,F11 24,F3 24,F23 25,F101 25,F23 25,F17
26,F101 26,F13 26,F11 27,F410 27,F19 27,F17
28,F410 28,F19 28,F11 29,F220 29,F17 29,F101 30,F220 30,F11 30,F410
31,F7 31,F19 31,F17 32,F7 32,F19 32,F11 33,F11 33,F410
34,F19 34,F17 34,F101 35,F23 35,F11 35,F410 36,F3 36,F23 36,F17
37,F3 37,F23 37,F11;

ENDDATA
!-----;
! Minimize cost of facilities opened, demands under or over served,;
MIN = @SUM( PATTERN( I): COST( I) * Y(I))
+ @SUM( DMND( J): CU( J) * U( J) + CV( J) * V( J));

! For each demand,
sum of patterns serving it + under variable - over variable= 1;
@FOR( DMND( J):
[COV] @SUM( PXD( I, J): Y( I)) + U( J) - V( J) = 1;
);

! Stay within budget on facilities cost;
@SUM( PATTERN: COST * Y) <= BUDGET;

! and demand under and overage costs;
@SUM( DMND: CU * U) <= BUDU;
@SUM( DMND: CV * V) <= BUDV;

! A PATTERN is either open or it is not, no halfsies;
@FOR( PATTERN( I): @BIN( Y( I)););

```

A solution obtained from this model is:

Variable	Value
Y(12)	1.000000
Y(24)	1.000000
Y(29)	1.000000
Y(32)	1.000000

Notice that it is different from our previous solution. It, however, also has a cost of 10(000), so it is an alternate optimum.

7.5 Problems

1. Certain types of facilities operate seven days each week and face the problem of allocating person power during the week as staffing requirements change as a function of the day of the week. This kind of problem is commonly encountered in public service and transportation organizations. Perhaps the most fundamental staffing problem involves the assignment of days off to full-time employees. In particular, it is regularly the case that each employee is entitled to two consecutive days off per week. If the number of employees required on each of the seven days of the week is given, then the problem is to find the minimum workforce size that will allow these demands to be met and then to determine the days off for the people in this workforce.

To be specific, let us study the problem faced by the Festus City Bus Company. The number of drivers required for each day of the week is as follows:

Mon	Tues	Wed	Thurs	Fri	Sat	Sun
18	16	15	16	19	14	12

How many drivers should be scheduled to start a five-day stint on each day of the week? Formulate this problem as a linear program. What is the optimal solution?

2. Completely unintentionally, several important details were omitted from the Festus City Staffing Problem (see previous question):
 - a) Daily pay is \$50 per person on weekdays, \$75 on Saturday, and \$90 on Sunday.
 - b) There are up to three people that can be hired who will work part-time, specifically, a 3-day week consisting of Friday, Sunday, and Monday. Their pay for this 3-day stint is \$200.

Modify the formulation appropriately. Is it obvious whether the part-time people will be used?

3. A political organization, Uncommon Result, wants to make a mass mailing to solicit funds. It has identified six “audiences” it wishes to reach. There are eight mailing lists it can purchase in order to get the names of the people in each audience. Each mailing list covers only a portion of the audiences. This coverage is indicated in the table below:

Mailing List	Audiences						Cost
	M.D.	LL.D.	D.D.S.	Business Executive	Brick Layers	Plumbers	
1	Y	N	N	Y	N	N	\$5000
2	N	Y	Y	N	N	N	\$4000
3	N	Y	N	N	N	Y	\$6000
4	Y	N	N	N	N	Y	\$4750
5	N	N	N	Y	N	Y	\$5500
6	N	N	Y	N	N	N	\$3000
7	N	Y	N	N	Y	N	\$5750
8	Y	N	N	N	Y	N	\$5250

A “Y” indicates the mailing list contains essentially all the names in the audience. An “N” indicates that essentially no names in the audience are contained in the mailing list. The costs associated with purchasing and processing a mailing list are given in the far right column. No change in total costs is incurred if an audience is contained in several mailing lists.

Formulate a model that will minimize total costs while ensuring all audiences are reached. Which mailing lists should be purchased?

4. The Pap-Iris Company prints various types of advertising brochures for a wide range of customers. The raw material for these brochures is a special finish paper that comes in 50-inch width rolls. The 50-inch width costs \$10 per inch-roll (i.e., \$500/roll). A roll is 1,000 feet long. Currently, Pap-Iris has three orders. Order number 1 is a brochure that is 16 inches wide with a run length of 400,000 feet. Order number 2 is a brochure that is 30 inches wide and has a run length of 80,000 feet. Order number 3 is a brochure that is 24 inches wide and has a run length of 120,000 feet. The major question is how to slit the larger raw material rolls into widths suitable for the brochures. With the paper and energy shortages, Pap-Iris wants to be as efficient as possible in its use of paper. Formulate an appropriate LP.
5. *Postal Optimality Analysis* (Due to Gene Moore). As part of a modernization effort, the U.S. Postal Service decided to improve the handling and distribution of bulk mail (second-, third- and fourth-class non-preferential) in the Chicago area. As part of this goal, a new processing facility was proposed for the Chicago area. One part of this proposal was development of a low-cost operational plan for the staffing of this facility. The plan would recognize the widely fluctuating hourly volume characteristic of such a facility and would suggest a staffing pattern or patterns that would accomplish the dual objectives of processing all mail received in a day while having no idle time.

A bulk mail processing facility, as the name implies, performs the function of receiving, unpacking, weighing, sorting by destination, and shipping of mail designated as non-preferential, including second class (bulk rate), third class (parcel post), and fourth class (books). It is frequently designed as a single purpose structure and is typically located in or adjacent to the large metropolitan areas, which produce this type of mail in significant volume. Although the trend in such facilities has been increased utilization of automated equipment (including highly sophisticated handling and sorting devices), paid manpower continues to account for a substantial portion of total operating expense.

Mail is received by the facility in mailbags and in containers. Both of which are shipped in trucks. It is also received in tied and wrapped packages, which are sent directly to the facility on railroad flatcars. Receipts of mail by the facility tend to be cyclical on a predictable basis throughout the 24-hour working day, resulting in the build-up of an “inventory” of mail during busy hours, which must be processed during less busy hours. A policy decision to have “no idle time” imposes a constraint on the optimal level of staffing.

Once the facility is ready for operations, it will be necessary to implement an operating plan that includes staffing requirements. A number of assumptions regarding such a plan are necessary at the outset. Some of them are based upon existing Postal Service policy, whereas others evolve from functional constraints. These assumptions are as follows:

- i. Pieces of mail are homogeneous in terms of processing effort.
- ii. Each employee can process 1800 pieces per hour.
- iii. Only full shifts are worked (i.e., it is impossible to introduce additional labor inputs or reduce existing labor inputs at times other than shift changes).
- iv. Shift changes occur at midnight, 8:00 A.M. and 4:00 P.M. (i.e., at the ends of the first, second and third shifts, respectively).
- v. All mail arrivals occur on the hour.
- vi. All mail must be processed the same day it is received (i.e., there may be no “inventory” carryover from the third shift to the following day’s first shift).
- vii. Labor rates, including shift differential, are given in the following table:

Shift	\$/Hour	Daily Rate
1st (Midnight-8 A.M.)	7.80	62.40
2nd (8 A.M. -4 P.M.)	7.20	57.60
3rd (4 P.M. -Midnight)	7.60	60.80

- viii. Hourly mail arrival is predictable and is given in the following table.

Cumulative Mail Arrival					
1st Shift		2nd Shift		3rd Shift	
Hour	Pieces	Hour	Pieces	Hour	Pieces
0100	56,350	0900	242,550	1700	578,100
0200	83,300	1000	245,000	1800	592,800
0300	147,000	1100	249,900	1900	597,700
0400	171,500	1200	259,700	2000	901,500
0500	188,650	1300	323,400	2100	908,850
0600	193,550	1400	369,950	2200	928,450
0700	210,700	1500	421,400	2300	950,500
0800	220,500	1600	485,100	2400	974,000

- a) Formulate the appropriate LP under the no idle time requirement. Can you predict beforehand the number to staff on the first shift? Will there always be a feasible solution to this problem for arbitrary arrival patterns?
- b) Suppose we allow idle time to occur. What is the appropriate formulation? Do you expect this solution to incur higher cost because it has idle time?

6. In the famous Northeast Tollway staffing problem, it was implied that at least a certain specified number of collectors were needed each period of the day. No extra benefit was assumed from having more collectors on duty than specified. You may recall that, because of the fluctuations in requirements over the course of the day, the optimal solution did have more collectors than required on duty during certain periods.

In reality, if more collectors are on duty than specified, the extra collectors are not completely valueless. The presence of the extra collectors results in less waiting for the motorists (boaters?). Similarly, if less than the specified number of collectors is on duty, the situation is not necessarily intolerable. Motorists will still be processed, but they may have to wait longer.

After much soul searching and economic analysis, you have concluded that one full-time collector costs \$100 per shift. An extra collector on duty for one hour results in \$10 worth of benefits to motorists. Further, having one less collector on duty than required during an one-hour period results in a waiting cost to motorists of \$50.

Assuming you wish to minimize total costs (motorists and collectors), show how you would modify the LP formulation. You only need illustrate for one constraint.

7. Some cities have analyzed their street cleaning and snow removal scheduling by methods somewhat analogous to those described for the Sayre-Priors airline problem. After a snowfall, each of specified streets must be swept by at least one truck.
- What are the analogs of the flight legs in Sayre-Priors?
 - What might be the decision variables corresponding to the tours in Sayre-Priors?
 - Should the constraints be equality or inequality in this case and why?
8. The St. Libory Quarry Company (SLQC) sells the rock that it quarries in four grades: limestone, chat, Redi-Mix-Grade, and coarse. A situation it regularly encounters is one in which it has large inventories of the grades it does not need and very little inventory in the grades needed at the moment. Large rocks removed from the earth are processed through a crusher to produce the four grades. For example, this week it appears the demand is for 50 tons of limestone, 60 tons of chat, 70 tons of Redi-Mix, and 30 tons of coarse. Its on-hand inventories for these same grades are respectively: 5, 40, 30, and 40 tons. For practical purposes, one can think of the crusher as having three operating modes: close, medium, and coarse. SLQC has gathered some data and has concluded one ton of quarried rock gets converted into the following output grades according to the crusher setting as follows:

Crusher Operating Mode	Tons Output per Ton Input				Operating Cost/Ton
	Limestone	Chat	Redi-Mix	Coarse	
Close	0.50	0.30	0.20	0.00	\$8
Medium	0.20	0.40	0.30	0.10	\$5
Coarse	0.05	0.20	0.35	0.40	\$3

SLQC would like to know how to operate its crusher to bring inventories up to the equivalent of at least two weeks worth of demand. Provide whatever help your current circumstances permit.

9. A certain optical instrument is being designed to selectively provide radiation over the spectrum from about 3500 to 6400 Angstrom units. To cover this optical range, a range of chemicals must be incorporated into the design. Each chemical provides coverage of a certain range. A list of the available chemicals, the range each covers, and its relative cost is provided below.

Chemical	Range Covered in Angstroms		Relative Cost
	Lower Limit	Upper Limit	
PBD	3500	3655	4
PPO	3520	3905	3
PPF	3600	3658	1
PBO	3650	4075	4
PPD	3660	3915	1
POPOP	3900	4449	6
A-NPO	3910	4095	2
NASAL	3950	4160	3
AMINOB	3995	4065	1
BBO	4000	4195	2
D-STILB	4000	4200	2
D-POPOP	4210	4405	2
A-NOPON	4320	4451	2
D-ANTH	4350	4500	2
4-METHYL-V	4420	5400	9
7-D-4-M	4450	4800	3
ESCULIN	4450	4570	1
NA-FLUOR	5200	6000	9
RHODAMINE-6G	5600	6200	8
RHODAMINE-B	6010	6400	8
ACRIDINE-RED	6015	6250	2

What subset of the available chemicals should be chosen to provide uninterrupted coverage from 3500 to 6400 Angstroms?

10. A manufacturer has the following orders in hand:

Order Units	X	Y	Z
60,000	90,000	300,000	
.45	.24	.16	

Each order is for a single distinct type of product. The manufacturer has four different production processes available for satisfying these orders. Each process produces a different combination of the three products. Each process costs \$0.50 per unit. The manufacturer must satisfy the volumes specified in the above table. The manufacturer formulated the following LP:

$$\begin{aligned} \text{Min } & .5 A + .5 B + .5 C + .5 D \\ \text{s.t. } & A && \geq 60000 \\ & 2B + C && \geq 90000 \\ & A + C + 3D && \geq 300000 \end{aligned}$$

- a) Which products are produced by process C?
- b) Suppose the agreements with customers are such that, for each product, the manufacturer is said to have filled the order if the manufacturer delivers an amount within + or - 10% of the "nominal" volume in the above table. The customer pays for whatever is delivered. Modify the formulation to incorporate this more flexible arrangement.
11. The formulation and solution of a certain staff-scheduling problem are shown below:

```

MIN M + T + W + R + F + S + N
      T + W + R + F + S      >= 14
      W + R + F + S + N      >= 9
      M          + R + F + S + N      >= 8
      M + T          + F + S + N      >= 6
      M + T + W          + S + N      >= 17
      M + T + W + R          + N      >= 15
      M + T + W + R + F          >= 18
END

Optimal solution found at step:        4
Objective value: 19.0000000
Variable      Value      Reduced Cost
M            5.000000      .0000000
T            .0000000      .0000000
W            11.00000      .0000000
R            .0000000      .0000000
F            2.000000      .0000000
S            1.000000      .0000000
N            .0000000      .3333333

Row    Slack or Surplus      Dual Price
2      .0000000      -.3333333
3      5.000000      .0000000
4      .0000000      -.3333333
5      2.000000      .0000000
6      .0000000      -.3333333
7      1.000000      .0000000
8      .0000000      -.3333333

```

where, M, T, W, R, F, S, N is the number of people starting their five-day work week on Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday respectively.

- a) How many people are required on duty on Thursday?
 - b) Suppose that part-time helpers are available who will work the three-day pattern, Thursday, Friday, and Saturday. That is, if you hire one of them, they will work all three days. These people cost 20% more per day than the ordinary folk who work a five-day week. Let P denote the number of part-timers to hire. Show how to modify the formulation to incorporate this option.
 - c) Using information from the solution report, what can you say about the (economic) attractiveness of the part-time help?
12. Acie Knielson runs a small survey research company out of a little office on the Northwest side. He has recently been contracted to do a telephone survey of the head-of-household of at least 220 households. The demographics of the survey must satisfy the following profile:

Age of head-of-household	18-25	26-35	36-60	≥ 61
Households in survey (min):	30	50	100	40

When Acie makes a phone call, he knows only on average what kind of head-of-household he will find (if any). Acie can make either daytime or nighttime phone calls. Calls at night have a higher probability of success. However, they cost more because a higher wage must be paid. Being a surveyor, Acie has good statistics on all this. Specifically, from past experience, he knows he can expect:

Call Type	Percent of calls finding head-of-household of given type					Cost/call
	18-25	26-35	36-60	61 or more	Not at home	
Day	2%	2%	8%	15%	73%	\$2.50
Night	4%	14%	28%	18%	36%	\$5.50

In words, what are the decision variables? What are the constraints? What is your recommendation? How much do you estimate this project will cost Acie?

13. A political candidate wants to make a mass mailing of some literature to counteract some nasty remarks his opponent has recently made. Five mailing lists have been identified that contain names and addresses of voters our candidate might like to reach. Each list may be purchased for a price. A partial list cannot be purchased. The numbers of names each list contains in each of four professions are listed below.

Mailing List	Names on Each List (in 1000's) by Profession				Cost of List
	Law	Health	Business Executives	Craft Professionals	
1	28	4	7	2	41,000
2	9	29	11	3	52,000
3	6	3	34	18	61,000
4	2	4	6	20	32,000
5	8	9	12	14	43,000
Desired Coverage	20	18	22	20	

Our candidate has estimated how many voters he wants to reach in each profession. This is listed in the row “Desired Coverage”. Having a more limited budget than the opponent, our candidate does not want to spend any more than he has to in order to “do the job”.

- a) How many decision variables would you need to model this problem?
 - b) How many constraints would you need to model this problem?
 - c) Define the decision variables you would use and write the objective function.
 - d) Write a complete model formulation.
14. Your agency provides telephone consultation to the public from 7 a.m. to 8 p.m., five days a week. The telephone load on your agency is heaviest in the months around April 15 of each year. You would like to set up staffing procedures for handling this load during these busy months. Each telephone consultant you hire starts work each day at either 7, 8, 9, 10, or 11 a.m., works for four hours, is off for one hour, and then works for another four hours. A complication that has become more noteworthy in recent years is that an increasing fraction of the calls handled by your agency is from Spanish-speaking clients. Therefore, you must have some consultants who speak Spanish. You are able to hire two kinds of consultants: English-speaking only, and bilingual (i.e., both English- and Spanish-speaking). A bilingual consultant can handle English and Spanish calls equally well. It should not be surprising that a bilingual consultant costs 1.1 times as much as an English-only consultant. You have collected some data on the call load by hour of the day and language type, measured in consultants required, for one of your more important offices. These data are summarized below:

Hour of the day:	7	8	9	10	11	12	1	2	3	4	5	6	7
English load:	4	4	5	6	6	8	5	4	4	5	5	5	3
Spanish load:	5	5	4	3	2	3	4	3	2	1	3	4	4

For example, during the hour from 10 a.m. to 11a.m., you must have working at least three Spanish-speaking consultants plus at least six more who can speak English.

How many consultants of each type would you start at each hour of the day?

15. The well-known mail order company R. R. Bean staffs its order-taking phone lines seven days per week. Each staffer costs \$100 per day and can work five consecutive days per week. An important question is: Which five-day interval should each staffer work? One of the staffers is pursuing an MBA degree part-time and, as part of her coursework, developed the following model specific to R. R. Bean's staffing requirements.

```

MIN=500 * M + 500 * T + 500 * W + 500 * R + 500 * F
      + 500 * S + 500 * N;
[R1] M           + R     + F     + S     + N >= 6 ;
[R2] M   + T       + F     + S     + N >= 7 ;
[R3] M   + T   + W       + S     + N >= 11;
[R4] M   + T   + W   + R       + N >= 9 ;
[R5] M   + T   + W   + R   + F       >= 11;
[R6]      T   + W   + R   + F   + S       >= 9;
[R7]          W   + R   + F   + S   + N >= 10;
END

```

Note that M denotes the number of staffers starting on Monday, T the number starting on Tuesday, etc. $R1, R2$, etc., simply serve as row identifiers.

- What is the required staffing level on Wednesday (not the number to hire starting on Wednesday, which is a harder question)?
- Suppose you can hire people on a 3-day-per-week part-time schedule to work the pattern consisting of the three consecutive days, Wednesday, Thursday, Friday. Because of training, turnover, and productivity considerations of part-timers, you figure the daily cost of these part-timers will be \$105 per day. Show how this additional option would be added to the model above.
- Do you think the part-time option above might be worth using?
- When the above staffing requirements are met, there will nevertheless be some customer calls that are lost, because of the chance all staffers may be busy when a prospective customer calls. A fellow from marketing who is an expert on customer behavior and knows a bit of queuing theory estimates having an additional staffer on duty on any given day beyond the minimum specified in the model above is worth \$75. More than one above the minimum is of no additional value. For example, if the minimum number of staffers required on a day is 8, but there are actually 10 on duty, then the better service will generate \$75 of additional revenue. A third fellow, who is working on an economics degree part-time at Freeport Community College, argues that, because the \$75 per day benefit is less than the \$100 per day cost of a staffer, the solution will be unaffected by the \$75 consideration. Is this fellow's argument correct?
- To double check your answer to (b), you decide to generalize the formulation to incorporate the \$75 benefit of one-person overstaffing. Define any additional decision variables needed and show (i) any modifications to the objective function and to existing constraints and (ii) any additional constraints. You need only illustrate for one day of the week.

16. In a typical state lottery, a player submits an entry by choosing six numbers without replacement (i.e., no duplicates) from the set of numbers $\{1, 2, 3, \dots, 44\}$. After all entries have been submitted, the state randomly chooses six numbers without replacement from the set $\{1, 2, 3, \dots, 44\}$. If all of your six numbers match those of the state, then you win a big prize. If only five out of six of your numbers match those of the state, then you win a medium size prize. If only four out of six of your numbers match those of the state, then you win a small prize. If several winners chose the same set of numbers, then the prize is split equally among them. You are thinking of submitting two entries to the next state lottery. A mathematician friend suggests the two entries: $\{2, 7, 1, 8, 28, 18\}$ and $\{3, 1, 4, 15, 9, 2\}$. Another friend suggests simply $\{1, 2, 3, 4, 5, 6\}$ and $\{7, 8, 9, 10, 11, 12\}$. Which pair of entries has the higher probability of winning some prize?

17. For most cutting stock problems, the continuous LP solution is close to the IP solution. In particular, you may notice that many cutting stock problems possess the “integer round-up” feature. For example, if the LP solution requires 11.6 sets, then that is a good indication that there is an IP solution that requires exactly 12 sets. Does this round-up feature hold in general? Consider for example a problem in which there is a single raw material of width 600 cm. There are three finished good widths: 300 cm, 200 cm, and 120 cm, with requirements respectively of 3, 5, and 9 units.

8

Networks, Distribution and PERT/CPM

8.1 What's Special About Network Models

A subclass of models called network LPs warrants special attention for three reasons:

1. They can be completely described by simple, easily understood graphical figures.
2. Under typical conditions, they have naturally integer answers, and one may find a network LP a useful device for describing and analyzing the various shipment strategies.
3. They are frequently easier to solve than general LPs.

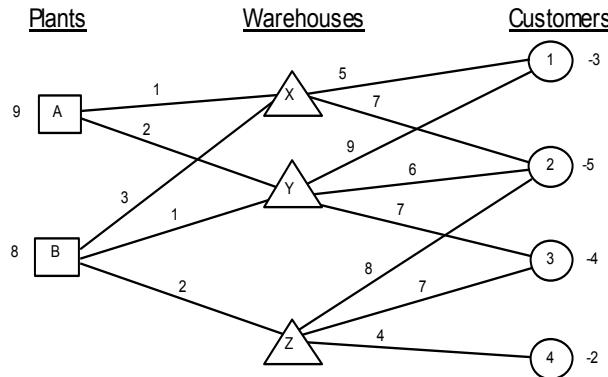
Physical examples that come to mind are pipeline or electrical transmission line networks. Any enterprise producing a product at several locations and distributing it to many warehouses and/or customers may find a network LP a useful device for describing and analyzing shipment strategies.

Although not essential, efficient specialized solution procedures may be used to solve network LPs. These procedures may be as much as 100 times faster than the general simplex method. Bradley, Brown, and Graves (1977) give a detailed description. Some of these specialized procedures were developed several years before the simplex method was developed for general LPs.

Figure 8.1 illustrates the network representing the distribution system of a firm using intermediate warehouses to distribute a product. The firm has two plants (denoted by *A* and *B*), three warehouses (denoted by *X*, *Y*, and *Z*), and four customer areas (denoted by 1, 2, 3, 4). The numbers adjacent to each node denote the availability of material at that node. Plant *A*, for example, has nine units available to be shipped. Customer 3, on the other hand, has -4 units meaning it needs to receive a shipment of four units.

The number above each arc is the cost per unit shipped along that arc. For example, if five of plant *A*'s nine units are shipped to warehouse *Y*, then a cost of $5 \times 2 = 10$ will be incurred as a direct result. The problem is to determine the amount shipped along each arc, so total costs are minimized and every customer has his requirements satisfied.

Figure 8.1 Three-Level Distribution Network



The essential condition on an LP for it to be a network problem is that it be representable as a network. There can be more than three levels of nodes, any number of arcs between any two nodes, and upper and lower limits on the amount shipped along a given arc.

With variables defined in an obvious way, the general LP describing this problem is:

```
[COST] MIN = AX + 2 * AY + 3 * BX + BY + 2 * BZ + 5 * X1
      + 7 * X2 + 9 * Y1 + 6 * Y2 + 7 * Y3 + 8 * Z2 + 7 * Z3
      + 4 * Z4;
[A] AX + AY <= 9;
[B] BX + BY + BZ <= 8;
[X] - AX - BX + X1 + X2 = 0;
[Y] - AY - BY + Y1 + Y2 + Y3 = 0;
[Z] - BZ + Z2 + Z3 + Z4 = 0;
[C1] - X1 - Y1 = -3;
[C2] - X2 - Y2 - Z2 = -5;
[C3] - Y3 - Z3 = -4;
[C4] - Z4 = -2;
```

There is one constraint for each node that is of a “sources = uses” form. Constraint 5, for example, is associated with warehouse Y and states that the amount shipped out minus the amount shipped in must equal 0.

A different view of the structure of a network problem is possible by displaying just the coefficients of the above constraints arranged by column and row. In the picture below, note that the apostrophes are placed every third row and column just to help see the regular patterns:

	A	A	B	B	B	X	X	Y	Y	Y	Z	Z	Z	
	X	Y	X	Y	Z	1	2	1	2	3	2	3	4	
COST:	1	2	3	1	2	5	7	9	6	7	8	7	4	MIN
A:	1	1	'	'	'	'	'	'	'	'	'	'	'	= 9
B:	'	'	1	1	1	'	'	'	'	'	'	'	'	= 8
X:	-1	'	-1	'	'	1	1	'	'	'	'	'	'	=
Y:	'	-1	'	-1	'	'	'	1	1	1	'	'	'	=
Z:	'	'	'	'	-1	'	'	'	'	'	1	1	1	=
C1:	'	'	'	'	'	'	'	'	'	'	'	'	'	= -3
C2:	'	'	'	'	'	'	'	-1	-1	-1	'	'	'	= -5
C3:	'	'	'	'	'	'	'	'	'	'	-1	'	'	= -4
C4:	'	'	'	'	'	'	'	'	'	'	'	-1	'	= -2

You should notice the key feature of the constraint matrix of a network problem. That is, without regard to any bound constraints on individual variables, each column has exactly two nonzeros in the constraint matrix. One of these nonzeros is a +1, whereas the other is a -1. According to the convention we have adopted, the +1 appears in the row of the node from which the arc takes material, whereas the row of the node to which the arc delivers material is a -1. On a problem of this size, you should be able to deduce the optimal solution manually simply from examining Figure 8.1. You may check it with the computer solution below:

Variable	Value	Reduced Cost
AX	3.000000	0.000000
AY	3.000000	0.000000
BX	0.000000	3.000000
BY	6.000000	0.000000
BZ	2.000000	0.000000
X1	3.000000	0.000000
X2	0.000000	0.000000
Y1	0.000000	5.000000
Y2	5.000000	0.000000
Y3	4.000000	0.000000
Z2	0.000000	3.000000
Z3	0.000000	1.000000
Z4	2.000000	0.000000
Row	Slack or Surplus	Dual Price
COST	100.000000	-1.000000
A	3.000000	0.000000
B	0.000000	1.000000
X	0.000000	1.000000
Y	0.000000	2.000000
Z	0.000000	3.000000
C1	0.000000	6.000000
C2	0.000000	8.000000
C3	0.000000	9.000000
C4	0.000000	7.000000

This solution exhibits two pleasing features found in the solution to any network problem:

1. If the right-hand side coefficients (the capacities and requirements) are integer, then the variables will also be integer.
2. If the objective coefficients are integer, then the dual prices will also be integer.

We can summarize network LPs as follows:

1. Associated with each node is a number that specifies the amount of commodity available at that node (negative implies that commodity is required.)
2. Associated with each arc are:
 - a cost per unit shipped (which may be negative) over the arc,
 - a lower bound on the amount shipped over the arc (typically zero), and
 - c) an upper bound on the amount shipped over the arc (infinity in our example).

The problem is to determine the flows that minimize total cost subject to satisfying all the supply, demand, and flow constraints.

8.1.1 Special Cases

There are a number of common applications of LP models that are special cases of the standard network LP. The ones worthy of mention are:

1. *Transportation or distribution problems.* A two-level network problem, where all the nodes at the first level are suppliers, all the nodes at the second level are users, and the only arcs are from suppliers to users, is called a transportation, or distribution model.
2. *Shortest and longest path problems.* Suppose one is given the road network of the United States and wishes to find the shortest route from Bangor to San Diego. This is equivalent to a special case of a network or transshipment problem in which one unit of material is available at Bangor and one unit is required at San Diego. The cost of shipping over an arc is the length of the arc. Simple, fast procedures exist for solving this problem. An important first cousin of this problem, the longest route problem, arises in the analysis of PERT/CPM projects.
3. *The assignment problem.* A transportation problem in which the number of suppliers equals the number of customers, each supplier has one unit available, and each customer requires one unit, is called an assignment problem. An efficient, specialized procedure exists for its solution.
4. *Maximal flow.* Given a directed network with an upper bound on the flow on each arc, one wants to find the maximum that can be shipped through the network from some specified origin, or source node, to some other destination, or sink node. Applications might be to determine the rate at which a building can be evacuated or military material can be shipped to a distant trouble spot.

8.2 PERT/CPM Networks and LP

Program Evaluation and Review Technique (PERT) and Critical Path Method (CPM) are two closely related techniques for monitoring the progress of a large project. A key part of PERT/CPM is calculating the critical path. That is, identifying the subset of the activities that must be performed exactly as planned in order for the project to finish on time.

We will show that the calculation of the critical path is a very simple network LP problem, specifically, a longest path problem. You do not need this fact to efficiently calculate the critical path, but it is an interesting observation that becomes useful if you wish to examine a multitude of “crashing” options for accelerating a tardy project.

In the table below, we list the activities involved in the simple, but nontrivial, project of building a house. An activity cannot be started until all of its predecessors are finished:

Activity	Mnemonic	Activity Time	Predecessors (Mnemonic)
Dig Basement	DIG	3	—
Pour Foundation	FOUND	4	DIG
Pour Basement Floor	POURB	2	FOUND
Install Floor Joists	JOISTS	3	FOUND
Install Walls	WALLS	5	FOUND
Install Rafters	RAFTERS	3	WALLS, POURB
Install Flooring	FLOOR	4	JOISTS
Rough Interior	ROUGH	6	FLOOR
Install Roof	ROOF	7	RAFTERS
Finish Interior	FINISH	5	ROUGH, ROOF
Landscape	SCAPE	2	POURB, WALLS

In Figure 8.2, we show the so-called PERT (or activity-on-arrow) network for this project. We would like to calculate the minimum elapsed time to complete this project. Relative to this figure, the number of interest is simply the longest path from left to right in this figure. The project can be completed no sooner than the sum of the times of the successive activities on this path. Verify for yourself that the critical path consists of activities *DIG*, *FOUND*, *WALLS*, *RAFTERS*, *ROOF*, and *FINISH* and has length 27.

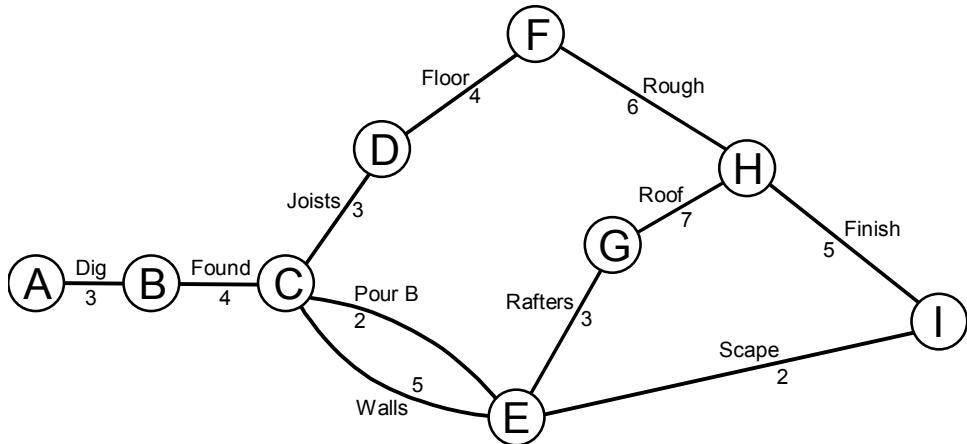
Even though this example can be worked out by hand, almost without pencil and paper, let us derive an LP formulation for solving this problem. Most people attempting this derivation will come up with one of two seemingly unrelated formulations.

The first formulation is motivated as follows. Let variables *DIG*, *FOUND*, etc. be either 1 or 0 depending upon whether activities *DIG*, *FOUND*, etc. are on or not on the critical path. The variables equal to one will define the critical path. The objective function will be related to the fact that we want to find the maximum length path in the PERT diagram.

Our objective is in fact:

$$\begin{aligned} \text{MAX} = & 3 * \text{DIG} + 4 * \text{FOUND} + 2 * \text{POURB} + 3 * \text{JOISTS} + \\ & 5 * \text{WALLS} + 3 * \text{RAFTERS} + 4 * \text{FLOOR} + 6 * \\ & \text{ROUGH} + 7 * \text{ROOF} + 5 * \text{FINISH} + 2 * \text{SCAPE}; \end{aligned}$$

Figure 8.2 Activity-on-Arc PERT/CPM Network



By itself, this objective seems to take the wrong point of view. We do not want to maximize the project length. However, if we specify the proper constraints, we shall see this objective will seek out the maximum length path in the PERT network. We want to use the constraints to enforce the following:

1. *DIG* must be on the critical path.
2. An activity can be on the critical path only if one of its predecessors is on the critical path. Further, if an activity is on a critical path, exactly one of its successors must be on the critical path, if it has successors.
3. Exactly one of *SCAPE* or *FINISH* must be on the critical path.

Convince yourself the following set of constraints will enforce the above:

- $\text{DIG} = -1;$
- $\text{FOUND} + \text{DIG} = 0;$
- $\text{JOISTS} - \text{POURB} - \text{WALLS} + \text{FOUND} = 0;$
- $\text{FLOOR} + \text{JOISTS} = 0;$
- $\text{RAFTERS} - \text{SCAPE} + \text{POURB} + \text{WALLS} = 0;$
- $\text{ROUGH} + \text{FLOOR} = 0;$
- $\text{ROOF} + \text{RAFTERS} = 0;$
- $\text{FINISH} + \text{ROUGH} + \text{ROOF} = 0;$
- + $\text{FINISH} + \text{SCAPE} = +1;$

If you interpret the length of each arc in the network as the scenic beauty of the arc, then the formulation corresponds to finding the most scenic route by which to ship one unit from *A* to *I*.

The solution of the problem is:

Optimal solution found at step:	2	
Objective value:	27.00000	
Variable		
DIG	1.000000	Reduced Cost 0.0000000
FOUND	1.000000	0.0000000
POURB	0.0000000	3.000000
JOISTS	0.0000000	0.0000000
WALLS	1.000000	0.0000000
RAFTERS	1.000000	0.0000000
FLOOR	0.0000000	0.0000000
ROUGH	0.0000000	2.000000
ROOF	1.000000	0.0000000
FINISH	1.000000	0.0000000
SCAPE	0.0000000	13.00000
Row		Dual Price
1	27.00000	1.000000
2	0.0000000	6.000000
3	0.0000000	-9.000000
4	0.0000000	-5.000000
5	0.0000000	-2.000000
6	0.0000000	0.0000000
7	0.0000000	2.000000
8	0.0000000	3.000000
9	0.0000000	10.00000
10	0.0000000	15.00000

Notice the variables corresponding to the activities on the critical path have a value of 1. What is the solution if the first constraint, $-DIG = -1$, is deleted?

It is instructive to look at the PICTURE of this problem in the following figure:

	R											
	J					A					F	
	F	P	O	W	A	F	R	O	R	I	S	
	O	O	I	A	T	L	O	U	O	N	C	
D	U	U	S	L	E	O	U	O	I	A		
I	N	R	T	L	R	O	G	O	S	P		
G	D	B	S	S	S	R	H	F	H	E		
1:	3	4	2	3	5	3	4	6	7	5	2 MAX	
2:	-1			'			'				= -1	
3:	1	-1	'	'	'	'	'	'	'	'	=	
4:		1	-1	-1	-1						=	
5:			1			-1					=	
6:	'	'	1	'	1	-1	'	'	'	-1	=	
7:							1	-1			=	
8:							1		-1		=	
9:	'	'	'	'	'			1	1	-1	' = 1	
10:										1	1 = 1	

Notice that each variable has at most two coefficients in the constraints. When two, they are +1 and -1. This is the distinguishing feature of a network LP.

Now, let us look at the second possible formulation. The motivation for this formulation is to minimize the elapsed time of the project. To do this, realize that each node in the PERT network represents an event (e.g., as follows: A , start digging the basement; C , complete the foundation; and I , complete landscaping and finish interior).

Define variables A, B, C, \dots, H, I as the time at which these events occur. Our objective function is then:

```
MIN = I - A;
```

These event times are constrained by the fact that each event has to occur later than each of its preceding events, at least by the amount of any intervening activity. Thus, we get one constraint for each activity:

```
B - A >= 3;           ! DIG;
C - B >= 4;           ! FOUND;
E - C >= 2;
D - C >= 3;
E - C >= 5;
F - D >= 4;
G - E >= 3;
H - F >= 6;
H - G >= 7;
I - H >= 5;
I - E >= 2;
```

The solution to this problem is:

Optimal solution found at step:	0	
Objective value:	27.00000	
Variable	Value	Reduced Cost
I	27.00000	0.0000000
A	0.0000000	0.0000000
B	3.0000000	0.0000000
C	7.0000000	0.0000000
E	12.00000	0.0000000
D	10.00000	0.0000000
F	14.00000	0.0000000
G	15.00000	0.0000000
H	22.00000	0.0000000
Row	Slack or Surplus	Dual Price
1	27.00000	1.000000
2	0.0000000	-1.000000
3	0.0000000	-1.000000
4	3.000000	0.0000000
5	0.0000000	0.0000000
6	0.0000000	-1.000000
7	0.0000000	0.0000000
8	0.0000000	-1.000000
9	2.000000	0.0000000
10	0.0000000	-1.000000
11	0.0000000	-1.000000
12	13.00000	0.0000000

Notice that the objective function value equals the critical path length. We can indirectly identify the activities on the critical path by noting the constraints with nonzero dual prices. The activities corresponding to these constraints are on the critical path. This correspondence makes sense. The right-hand side of a constraint is the activity time. If we increase the time of an activity on the critical path, it should increase the project length and thus should have a nonzero dual price. What is the solution if the first variable, A , is deleted?

The PICTURE of the coefficient matrix for this problem follows:

	A	B	C	D	E	F	G	H	I	
1:-1										1 MIN
2:-1	1									> 3
3:	'	-1	'	'	'	'	'	'	'	> 4
4:		-1	'	1		'				> 2
5:		-1	1			'				> 3
6:	'	-1	'	1	'	'	'	'	'	> 5
7:			-1		1	'				> 4
8:				-1		1				> 3
9:	'				-1	'	1	'	'	> 6
10:						-1	1			> 7
11:							-1	1		> 5
12:	'			'	-1	'	'	'	'	> 2

Notice the PICTURE of this formulation is essentially the PICTURE of the previous formulation rotated ninety degrees. Even though these two formulations originally were seemingly unrelated, there is really an incestuous relationship between the two, a relationship that mathematicians politely refer to as duality.

8.3 Activity-on-Arc vs. Activity-on-Node Network Diagrams

Two conventions are used in practice for displaying project networks: (1) Activity-on-Arc (AOA) and (2) Activity-on-Node (AON). Our previous example used the AOA convention. The characteristics of the two are:

AON

- Each activity is represented by a node in the network.
- A precedence relationship between two activities is represented by an arc or link between the two.
- AON may be less error prone because it does not need dummy activities or arcs.

AOA

- Each activity is represented by an arc in the network.
- If activity X must precede activity Y , there are X leads into arc Y . The nodes thus represent events or “milestones” (e.g., “finished activity X ”). Dummy activities of zero length may be required to properly represent precedence relationships.
- AOA historically has been more popular, perhaps because of its similarity to Gantt charts used in scheduling.

An AON project with six activities is shown in Figure 8.3. The number next to each node is the duration of the activity. Activities A and B are the sources or start of the project. Activity F is the final activity. By inspection, you can discover that the longest path consists of activities A , C , E , and F . It

has a length of 29. The corresponding AOA network for the same project is shown in Figure 8.4. In the AOA network, we have enclosed the activity letters in circles above the associated arc. The unenclosed numbers below each arc are the durations of the activities. We have given the nodes, or milestones, arbitrary number designations enclosed in squares. Notice the dummy activity (the dotted arc) between nodes 3 and 4. This is because *a dummy activity will be required in an AOA diagram anytime that two activities (e.g., A and B) share some (e.g., activity D), but not all (e.g., activity C), successor activities.*

Figure 8.3 An Activity-on-Node Representation

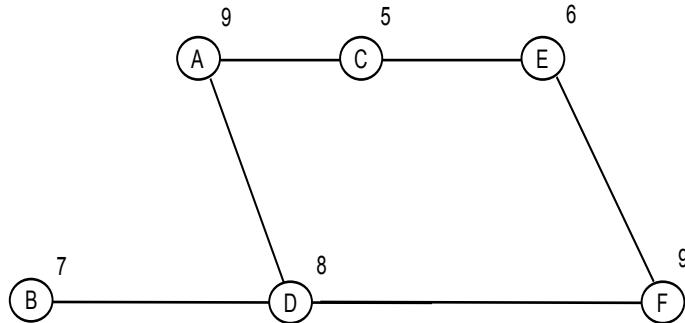
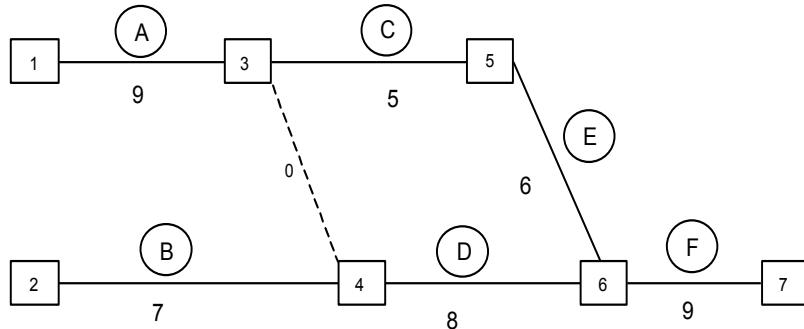


Figure 8.4 An Activity-on-Arc Representation



8.4 Crashing of Project Networks

Once the critical path length for a project has been identified, the next question invariably asked is: can we shorten the project? The process of decreasing the duration of a project or activity is commonly called crashing. For many construction projects, it is common for the customer to pay an incentive to the contractor for finishing the project in a shorter length of time. For example, in highway repair projects, it is not unusual to have incentives from \$5,000 to \$25,000 per day that the project is finished before a target date.

8.4.1 The Cost and Value of Crashing

There is value in crashing a project. In order to crash a project, we must crash one or more activities. Crashing an activity costs money. Deciding to crash an activity requires us to compare the cost of crashing that activity with the value of the resulting reduction in project length. This decision is frequently complicated by the fact that some negotiation may be required between the party that incurs the cost of crashing the activity (e.g., the contractor) and the party that enjoys the value of the crashed project (e.g., the customer).

8.4.2 The Cost of Crashing an Activity

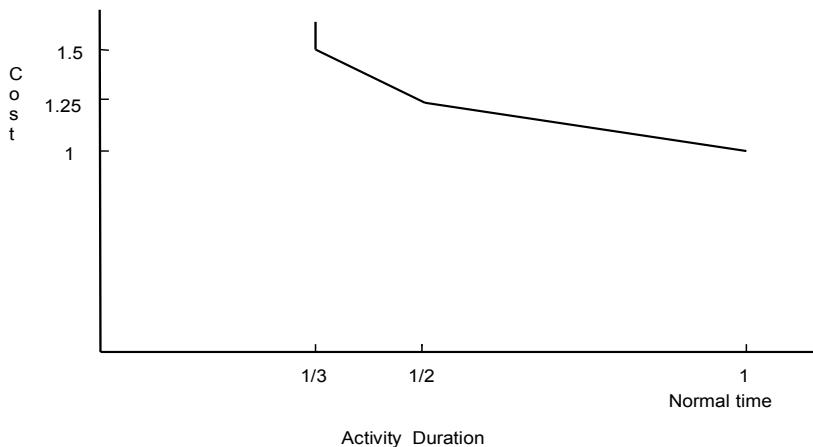
An activity is typically crashed by applying more labor to it (e.g., overtime or a second shift). We might typically expect that using second-shift labor could cost 1.5 times as much per hour as first-shift labor. We might expect third-shift labor to cost twice as much as first-shift labor.

Consider an activity that can be done in six days if only first-shift labor is used and has a labor cost of \$6,000. If we allow the use of second-shift labor and thus work two shifts per day, the activity can be done in three days for a cost of $3 \times 1000 + 3 \times 1000 \times 1.5 = 7,500$. If third-shift labor is allowed, then the project can be done in two days by working three shifts per day and incurring a total of:

$$2 \times 1000 + 2 \times 1000 \times 1.5 + 2 \times 1000 \times 2 = \$9,000.$$

Thus, we get a crashing cost curve for the activity as shown in Figure 8.5:

Figure 8.5 Activity Crash Cost Curve



8.4.3 The Value of Crashing a Project

There are two approaches to deciding upon the amount of project crashing: (a) we simply specify a project duration time and crash enough to achieve this duration, or (b) we estimate the value of crashing it for various days. As an example of (a), in 1987 a new stadium was being built for the Montreal Expos baseball team. The obvious completion target was the first home game of the season.

As an example of (b), consider an urban expressway repair. What is the value per day of completing it early? Suppose that 6,000 motorists are affected by the repair project and each is delayed by 10 minutes each day because of the repair work (e.g., by taking alternate routes or by slower traffic). The total daily delay is $6,000 \times 10 = 60,000$ minutes = 1000 hours. If we assign an hourly cost of \$5/person \times hours, the social value of reducing the repair project by one day is \$5,000.

8.4.4 Formulation of the Crashing Problem

Suppose we have investigated the crashing possibilities for each activity or task in our previous project example. These estimates are summarized in the following table:

Activity	Predecessor	Normal duration	Minimum duration	\$/Day
		(Days)	if crashed (Days)	
A	—	9	5	5000
B	—	7	3	6000
C	A	5	3	4000
D	A,B	8	4	2000
E	C	6	3	3000
F	D,E	9	5	9000

For example, activity *A* could be done in five days rather than nine. However, this would cost us an extra $(9 - 5) \times 5000 = \$20,000$.

First, consider the simple case where we have a hard due date by which the project must be done. Let us say 22 days in this case. How would we decide which activities to crash? Activity *D* is the cheapest to crash per day. However, it is not on the critical path, so its low cost is at best just interesting.

Let us define:

$$EF_i = \text{earliest finish time of activity } i, \text{ taking into account any crashing that is done}; \\ C_i = \text{number of days by which activity } i \text{ is crashed}.$$

In words, the LP model will be:

Minimize Cost of crashing

subject to

For each activity *j* and each predecessor *i*:

earliest finish of *j* \geq earliest finish of predecessor *i* + actual duration of *j*;

For each activity *j*:

minimum duration for *j* if crashed \leq actual duration of *j* \leq normal duration for *j*.

A LINGO formulation is:

```
! Find optimal crashing for a project with a due date;
SETS:
  TASK: NORMAL, FAST, COST, EF, ACTUAL,
  PRED( TASK, TASK) :;
ENDSETS
```

```

DATA:
  TASK, NORMAL, FAST, COST =
    A      9      5     5000
    B      7      3     6000
    C      5      3     4000
    D      8      4     2000
    E      6      3     3000
    F      9      5    9000;
  PRED =
    A, C
    A, D
    B, D
    C, E
    D, F
    E, F;
  DUEDATE = 22;
ENDDATA
!-----;
! Minimize the cost of crashing;
[OBJ] MIN = @SUM( TASK( I): COST( I)*( NORMAL( I) - ACTUAL( I)));
! For tasks with no predecessors...
@FOR( TASK( J): EF( J) >= ACTUAL( J));
! and for those with predecessors;
@FOR( PRED( I, J):
  EF( J) >= EF( I) + ACTUAL( J);
);
! Bound the actual time;
@FOR( TASK( I):
  @BND( FAST(I), ACTUAL( I), NORMAL( I));
);
! Last task is assumed to be last in project;
EF( @SIZE( TASK)) <= DUEDATE;

```

Part of the solution is:

Global optimal solution found at step: 24		
Objective value: 31000.00		
Variable	Value	Reduced Cost
EF(A)	7.000000	0.0000000
EF(B)	7.000000	0.0000000
EF(C)	10.000000	0.0000000
EF(D)	13.000000	0.0000000
EF(E)	13.000000	0.0000000
EF(F)	22.000000	0.0000000
ACTUAL(A)	7.000000	0.0000000
ACTUAL(B)	7.000000	-4000.000
ACTUAL(C)	3.000000	1000.000
ACTUAL(D)	6.000000	0.0000000
ACTUAL(E)	3.000000	2000.000
ACTUAL(F)	9.000000	-2000.000

Thus, for an additional cost of \$31,000, we can meet the 22-day deadline.

Now, suppose there is no hard project due date, but we do receive an incentive payment of \$5000 for each day we reduce the project length. Define $PCRASH$ = number of days the project is finished before the twenty-ninth day. Now, the formulation is:

```

! Find optimal crashing for a project with
    a due date and incentive for early completion;
SETS:
  TASK: NORMAL, FAST, COST, EF, ACTUAL;
  PRED( TASK, TASK):;
ENDSETS
DATA:
  TASK, NORMAL, FAST, COST =
    A      9      5      5000
    B      7      3      6000
    C      5      3      4000
    D      8      4      2000
    E      6      3      3000
    F      9      5      9000;
  PRED =
    A, C
    A, D
    B, D
    C, E
    D, F
    E, F;
! Incentive for each day we beat the due date;
  INCENT = 5000;
  DUEDATE = 29;
ENDDATA
!-----
! Minimize the cost of crashing
    less early completion incentive payment;
  [OBJ] MIN = @SUM( TASK( I): COST( I)*( NORMAL( I) - ACTUAL( I)))
    - INCENT * PCRASH;
! For tasks with no predecessors...
  @FOR( TASK( J): EF( J) >= ACTUAL( J););
! and for those with predecessors;
  @FOR( PRED( I, J):
    EF( J) >= EF( I) + ACTUAL( J);
  );
! Bound the actual time;
  @FOR( TASK( I):
    @BND( FAST(I), ACTUAL( I), NORMAL( I));
  );
! Last task is assumed to be last in project;
  EF( @SIZE( TASK)) + PCRASH = DUEDATE;

```

From the solution, we see we should crash it by five days to give a total project length of twenty-four days:

Global optimal solution found at step:	21	
Objective value:	-6000.000	
Variable	Value	Reduced Cost
PCRASH	5.000000	0.0000000
EF(A)	7.000000	0.0000000
EF(B)	7.000000	0.0000000
EF(C)	12.000000	0.0000000
EF(D)	15.000000	0.0000000
EF(E)	15.000000	0.0000000
EF(F)	24.000000	0.0000000
ACTUAL(A)	7.000000	0.0000000
ACTUAL(B)	7.000000	-6000.000
ACTUAL(C)	5.000000	-1000.000
ACTUAL(D)	8.000000	0.0000000
ACTUAL(E)	3.000000	0.0000000
ACTUAL(F)	9.000000	-4000.000

The excess of the incentive payments over crash costs is \$6,000.

8.5 Resource Constraints in Project Scheduling

For many projects, a major complication is that there are a limited number of resources. The limited resources require you to do tasks individually that otherwise might be done simultaneously. Pritzker, Watters, and Wolfe (1969) gave a formulation representing resource constraints in project and jobshop scheduling problems. The formulation is based on the following key ideas: a) time is discrete rather than continuous (e.g., each period is a day), b) for every activity and every discrete period there is a 0/1 variable that is one if that activity starts in that period, and c) for every resource and period there is a constraint that enforces the requirement that the amount of resource required in that period does not exceed the amount available.

To illustrate, we take the example considered previously with shorter activity times, so the total number of periods is smaller:

```

MODEL:
! PERT/CPM project scheduling with resource constraints(PERTRSRC);
! There is a limited number of each resource/machine.
! An activity cannot be started until: 1) all its predecessors have
completed, and 2) resources/machines required are available.;

SETS:
! There is a set of tasks with a given duration, and
! a start time to be determined;
TASK: TIME, START, ES;
! The precedence relations, the first task in the
precedence relationship needs to be completed before the
second task can be started;
PRED( TASK, TASK);
! There are a set of periods;
PERIOD;

```

```

    RESOURCE: CAP;
! Some operations need capacity in some department;
    TXR( TASK, RESOURCE): NEED;
! SX( I, T) = 1 if task I starts in period T;
    TXP( TASK, PERIOD): SX;
    RXP( RESOURCE, PERIOD);
ENDSETS

DATA:
! Upper limit on number of periods required to complete the project;
    PERIOD = 1..20;
! Task names and duration;
    TASK   TIME =
    FIRST      0
    FCAST      7
    SURVEY    2
    PRICE      1
    SCHEd     3
    COSTOUT   2
    FINAL      4;

! The predecessor/successor combinations;
    PRED=  FIRST,FCAST,      FIRST,SURVEY,
           FCAST,PRICE,      FCAST,SCHEd,      SURVEY,PRICE,
           SCHEd,COSTOUT,    PRICE,FINAL,      COSTOUT,FINAL;
! There are 2 departments, accounting and operations,
with capacities...;
    RESOURCE = ACDEPT, OPNDEPT;
    CAP = 1, 1;
! How much each task needs of each resource;
    TXR,      NEED =
    FCAST,  OPNDEPT, 1
    SURVEY, OPNDEPT, 1
    SCHEd,  OPNDEPT, 1
    PRICE,   ACDEPT, 1
    COSTOUT, ACDEPT, 1;
ENDDATA
!-----;
! Minimize start time of last task;
    MIN = START( @SIZE( TASK));
! Start time for each task;
@FOR( TASK( I):
    [DEFSTRT] START( I) = @SUM( PERIOD( T): T * SX( I, T));
    );
@FOR( TASK( I):
! Each task must be started in some period;
    [MUSTDO]  @SUM( PERIOD( T): SX( I, T)) = 1;
! The SX vars are binary, i.e., 0 or 1;
    @FOR( PERIOD( T): @BIN( SX( I, T)));
    );
! Precedence constraints;
@FOR( PRED( I, J):
    [PRECD]  START( J) >= START( I) + TIME( I);

```

```

    );
! Resource usage, For each resource R and period T;
@FOR( RXP( R, T):
! Sum over all tasks I that use resource R in period T;
[RSRUSE] @SUM( TXR( I, R):
    @SUM( PERIOD( S) | S #GE# ( T - ( TIME( I) - 1)) #AND# S #LE# T:
        NEED( I, R) * SX( I, S))) <= CAP( R);
);
! The following makes the formulation tighter;
! Compute earliest start disregarding resource constraints;
@FOR( TASK( J):
    ES( J) = @SMAX( 0, @MAX( PRED( I, J): ES( I) + TIME(I)));
    ! Task cannot start earlier than unconstrained early start;
    @SUM( PERIOD(T) | T #LE# ES( J): SX( J, T)) = 0;
);
END

```

When solved, we get a project length of 14. If there were no resource constraints, then the project length would be 13:

```

Global optimal solution found
Objective value:      14.00000
Variable          Value
START( FIRST)    1.000000
START( FCAST)    1.000000
START( SURVEY)   11.00000
START( PRICE)    13.00000
START( SCHEDE)   8.000000
START( COSTOUT)  11.00000
START( FINAL)    14.00000

```

8.6 Path Formulations

In many network problems, it is natural to think of a solution in terms of paths that material takes as it flows through the network. For example, in Figure 8.1, there are thirteen possible paths. Namely:

$A \rightarrow X \rightarrow 1, A \rightarrow X \rightarrow 2, A \rightarrow Y \rightarrow 1, A \rightarrow Y \rightarrow 2, A \rightarrow Y \rightarrow 3, B \rightarrow X \rightarrow 1, B \rightarrow X \rightarrow 2,$
 $B \rightarrow Y \rightarrow 1, B \rightarrow Y \rightarrow 2, B \rightarrow Y \rightarrow 3, B \rightarrow Z \rightarrow 2, B \rightarrow Z \rightarrow 3,$
 $B \rightarrow Z \rightarrow 4$

One can, in fact, formulate decision variables in terms of complete paths rather than just simple links, where the path decision variable corresponds to using a combination of links. This is a form of what is sometimes called a composite variable approach. The motivations for using the path approach are:

1. More complicated cost structures can be represented. For example, Geoffrion and Graves (1974) use the path formulation to represent “milling in transit” discount fare structures in shipping food and feed products.

2. Path-related restrictions can be incorporated. For example, regulations allow a truck driver to be on duty for at most 10 hours. Thus, in a truck routing network one would not consider paths longer than 10 hours. In a supply chain network, a path that is long may be prohibited because it may cause lead times to be too long.
3. The number of rows (constraints) in the model may be substantially less.
4. In integer programs where some, but not all, of the problem has a network structure, the path formulation may be easier to solve.

8.6.1 Example

Let us reconsider the first problem (Figure 8.1, page 146). Suppose shipments from A to X are made by the same carrier as shipments from X to 2. This carrier will give a \$1 per unit “milling-in-transit” discount for each unit it handles from both A to X and X to 2. Further, the product is somewhat fragile and cannot tolerate a lot of transportation. In particular, it cannot be shipped both over link $B \rightarrow X$ and $X \rightarrow 2$ or both over links $A \rightarrow Y$ and $Y \rightarrow 1$.

Using the notation AXI = number of units shipped from A to X to 1, etc., the path formulation is:

```

MIN = 6 * PAX1 + 7 * PAX2 + 8 * PAY2
      + 9 * PAY3 + 8 * PBX1 + 10 * PBY1
      + 7 * PBY2 + 8 * PBY3 + 10 * PBZ2
      + 9 * PBZ3 + 6 * PBZ4;

[A] PAX1 + PAX2 + PAY2 + PAY3 <= 9;
[B] PBX1 + PBY1 + PBY2 + PBY3
      + PBZ2 + PBZ3 + PBZ4 <= 8;
[C1] PAX1 + PBX1 + PBY1 = 3;
[C2] PAX2 + PAY2 + PBY2 + PBZ2 = 5;
[C3] PAY3 + PBY3 + PBZ3 = 4;
[C4] PBZ4 = 2;

```

Notice the cost of path $AX2 = 1 + 7 - 1 = 7$. In addition, paths $BX2$ and $AY1$ do not appear. This model has only six constraints as opposed to nine in the original formulation. The reduction in constraints arises from the fact that, in path formulations, one does not need the “sources = uses” constraints for intermediate nodes.

In general, the path formulation will have fewer rows, but more decision variables than the corresponding network LP model.

When we solve, we get:

Objective value=	97.0000
Variable	Value
PAX1	3.000000
PAX2	3.000000
PBY2	2.000000
PBY3	4.000000
PBZ4	2.000000

This is cheaper than the previous solution, because the three units shipped over path $AX2$ go for \$1 per unit less.

A path formulation need not have a naturally integer solution. If the path formulation, however, is equivalent to a network LP, then it will have a naturally integer solution.

The path formulation is popular in long-range forest planning. See, for example, Davis and Johnson (1986), where it is known as the “Model I” approach. The standard network LP based

formulation is known as the “Model II” approach. In a forest planning Model II, a link in the network represents a decision to plant an acre of a particular kind of tree in some specified year and harvest it in some future specified year. A node represents a specific harvest and replant decision. A decision variable in Model I is a complete prescription of how to manage (i.e., harvest and replant) a given piece of land over time. Some Model I formulations in forest planning may have just a few hundred constraints, but over a million decision variables or paths.

There is a generalization of the path formulation to arbitrary linear programs, known as Fourier/Motzkin/Dines elimination, see for example Martin (1999) and Dantzig (1963). The transformation of a network LP to the path formulation involves eliminating a particular node (constraint), by generating a new variable for every combination of input arc and output arc incident to the node. A constraint in an arbitrary LP can be eliminated if it is first converted to a constraint with a right-hand side of zero and then a new variable is generated for every combination of positive and negative coefficient in the constraint. The disadvantage of this approach is that even though the number of constraints is reduced to one, the number of variables may grow exponentially with the number of original constraints.

A variable corresponding to a path in a network is an example of a composite variable, a general approach that is sometimes useful for representing complicated/ing constraints. A composite variable is one that represents a feasible combination of two or more original variables. The complicating constraints are represented implicitly by generating only those composite variables that correspond to feasible combinations and values of the original variables.

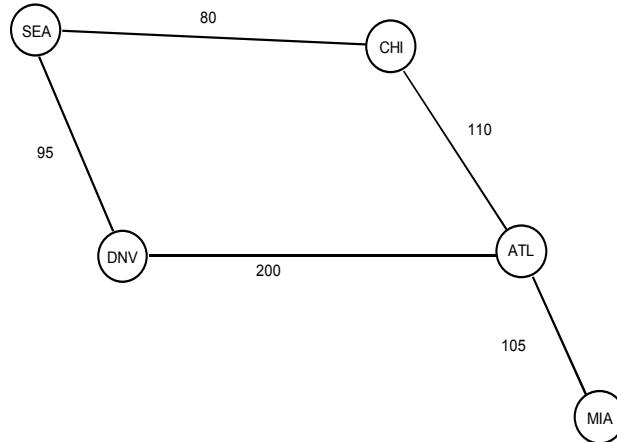
8.7 Path Formulations of Undirected Networks

In many communications networks, the arcs have capacity, but are undirected. For example, when you are carrying on a phone conversation with someone in a distant city, the conversation uses capacity on all the links in your connection. However, you cannot speak of a direction of flow of the connection.

A major concern for a long distance communications company is the management of its communications network. This becomes particularly important during certain holidays, such as Mother’s Day. Not only does the volume of calls increase on these days, but also the pattern of calls changes dramatically from the business-oriented traffic during weekdays in the rest of the year. A communications company faces two problems: (a) the design problem. That is, what capacity should be installed on each link? As well as, (b) the operations problem. That is, given the installed capacity, how are demands best routed? The path formulation is an obvious format for modeling an undirected network. The following illustrates the operational problem.

Consider the case of a phone company with the network structure shown in Figure 8.6:

Figure 8.6 Phone Company Network Structure



The number next to each arc is the number of calls that can be in progress simultaneously along that arc. If someone in *MIA* tries to call his mother in *SEA*, the phone company must first find a path from *MIA* to *SEA* such that each arc on that path is not at capacity. It is quite easy to inefficiently use the capacity. Suppose there is a demand for 110 calls between *CHI* and *DNV* and 90 calls between *ATL* and *SEA*. Further, suppose all of these calls were routed over the *ATL*, *DNV* link. Now, suppose we wish to make a call between *MIA* and *SEA*. Such a connection is impossible because every path between the two contains a saturated link (i.e., either *ATL*, *DNV* or *CHI*, *ATL*). However, if some of the 110 calls between *CHI* and *DNV* were routed over the *CHI*, *SEA*, *DNV* links, then one could make calls between *MIA* and *SEA*. In conventional voice networks, a call cannot be rerouted once it has started. In packet switched data networks and, to some extent, in cellular phone networks, some rerouting is possible.

8.7.1 Example

Suppose during a certain time period the demands in the table below occur for connections between pairs of cities:

	DNV	CHI	ATL	MIA
SEA	10	20	38	33
DNV		42	48	23
CHI			90	36
ATL				26

Which demands should be satisfied and via what routes to maximize the number of connections satisfied?

Solution. If we use the path formulation, there will be two paths between every pair of cities except *ATL* and *MIA*. We will use the notation P_{1ij} for number of calls using the shorter or more northerly path between cities i and j , and P_{2ij} for the other path, if any. There will be two kinds of constraints:

- 1) a capacity constraint for each link, and
- 2) an upper limit on the calls between each pair of cities, based on available demand.

A formulation is:

```

! Maximize calls carried;
MAX = P1MIAATL + P1MIADNV + P2MIADNV
      + P1MIASEA + P2MIASEA + P1MIACHI
      + P2MIACHI + P1ATLDNV + P2ATLDNV
      + P1ATLSEA + P2ATLSEA + P1ATLCHI
      + P2ATLCHI + P1DNVSEA + P2DNVSEA
      + P1DNVCHI + P2DNVCHI + P1SEACHI
      + P2SEACHI;

! Capacity constraint for each link;
[KATLMIA]  P1MIAATL + P1MIADNV + P2MIADNV
            + P1MIASEA + P2MIASEA + P1MIACHI
            + P2MIACHI <= 105;
[KATLDNV]  P1MIADNV + P1MIASEA + P1MIACHI
            + P1ATLDNV + P1ATLSEA + P1ATLCHI
            + P2DNVSEA + P2DNVCHI + P2SEACHI <= 200;
[KDNVSEA]  P2MIADNV + P1MIASEA + P1MIACHI
            + P2ATLDNV + P1ATLSEA + P1ATLCHI
            + P1DNVSEA + P1DNVCHI + P2SEACHI <= 95;
[KSEACHI]  P2MIADNV + P2MIASEA + P1MIACHI
            + P2ATLDNV + P2ATLSEA + P1ATLCHI
            + P2DNVSEA + P1DNVCHI + P1SEACHI <= 80;
[KATLCHI]  P2MIADNV + P2MIASEA + P2MIACHI
            + P2ATLDNV + P2ATLSEA + P2ATLCHI
            + P2DNVSEA + P2DNVCHI + P2SEACHI <= 110;
! Demand constraints for each city pair;
[DMIATL]    P1MIAATL <= 26;
[DMIADNV]   P1MIADNV + P2MIADNV <= 23;
[DMIASEA]   P1MIASEA + P2MIASEA <= 33;
[DMIACHI]   P1MIACHI + P2MIACHI <= 36;
[DATLDNV]   P1ATLDNV + P2ATLDNV <= 48;
[DATLSEA]   P1ATLSEA + P2ATLSEA <= 38;
[DATLCHI]   P1ATLCHI + P2ATLCHI <= 90;
[DDNVSEA]   P1DNVSEA + P2DNVSEA <= 10;
[DDNVCHI]   P1DNVCHI + P2DNVCHI <= 42;
[DSEACHI]   P1SEACHI + P2SEACHI <= 20;

```

When this formulation is solved, we see we can handle 322 out of the total demand of 366 calls:

Optimal solution found at step:	11	
Objective value:	322.0000	
Variable	Value	Reduced Cost
P1MIAATL	26.00000	0.000000
P1MIADNV	23.00000	0.000000
P2MIADNV	0.00000	2.000000
P1MIASEA	0.00000	0.000000
P2MIASEA	0.00000	0.000000
P1MIACHI	25.00000	0.000000
P2MIACHI	0.00000	0.000000
P1ATLDNV	48.00000	0.000000
P2ATLDNV	0.00000	2.000000
P1ATLSEA	38.00000	0.000000
P2ATLSEA	0.00000	0.000000
P1ATLCHI	23.00000	0.000000
P2ATLCHI	67.00000	0.000000
P1DNVSEA	3.50000	0.000000
P2DNVSEA	6.50000	0.000000
P1DNVCHI	5.50000	0.000000
P2DNVCHI	36.50000	0.000000
P1SEACHI	20.00000	0.000000
P2SEACHI	0.00000	2.000000
Row	Slack or Surplus	Dual Price
1	322.00000	1.000000
KATLMIA	31.00000	0.000000
KATLDNV	0.00000	0.000000
KDNVSEA	0.00000	1.000000
KSEACHI	0.00000	0.000000
KATLCHI	0.00000	1.000000
DMIAATL	0.00000	1.000000
DMIADNV	0.00000	1.000000
DMIASEA	33.00000	0.000000
DMIACHI	11.00000	0.000000
DATLDNV	0.00000	1.000000
DATLSEA	0.00000	0.000000
DATLCHI	0.00000	0.000000
DDNVSEA	0.00000	0.000000
DDNVCHI	0.00000	0.000000
DSEACHI	0.00000	1.000000

Verify that the demand not carried is *MIA-CHI*: 11 and *MIA-SEA*: 33. Apparently, there are a number of alternate optima.

8.8 Double Entry Bookkeeping: A Network Model of the Firm

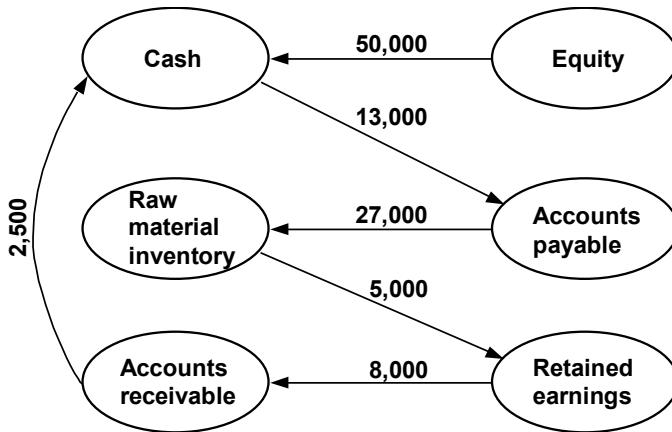
Authors frequently like to identify who was the first to use a given methodology. A contender for the distinction of formulating the first network model is Fra Luca Pacioli. In 1594, while director of a Franciscan monastery in Italy, he published a description of the accounting convention that has come to be known as double entry bookkeeping. From the perspective of networks, each double entry is an arc in a network.

To illustrate, suppose you start up a small dry goods business. During the first two weeks, the following transactions occur:

- | | |
|------------|--|
| <i>CAP</i> | 1) You invest \$50,000 of capital in cash to start the business. |
| <i>UR</i> | 2) You purchase \$27,000 of product on credit from supplier <i>S</i> . |
| <i>PAY</i> | 3) You pay \$13,000 of your accounts payable to supplier <i>S</i> . |
| <i>SEL</i> | 4) You sell \$5,000 of product to customer <i>C</i> for \$8,000 on credit. |
| <i>REC</i> | 5) Customer <i>C</i> pays you \$2,500 of his debt to you. |

In our convention, liabilities and equities will typically have negative balances. For example, the initial infusion of cash corresponds to a transfer (an arc) from the equity account (node) to the cash account, with a flow of \$50,000. The purchase of product on credit corresponds to an arc from the accounts payable account node to the raw materials inventory account, with a flow of \$27,000. Paying \$13,000 to the supplier corresponds to an arc from the cash account to the accounts payable account, with a flow of \$13,000. Figure 8.7 illustrates.

Figure 8.7 Double Entry Bookkeeping as a Network Model



8.9 Extensions of Network LP Models

There are several generalizations of network models that are important in practice. These extensions share two features in common with true network LP models, namely:

- They can be represented graphically.
- Specialized, fast solution procedures exist for several of these generalizations.

The one feature typically not found with these generalizations is:

- Solutions are usually not naturally integer, even if the input data are integers.

The important generalizations we will consider are:

1. *Networks with Gains.* Sometimes called generalized networks, this generalization allows a specified gain or loss of material as it is shipped from one node to another. Structurally, these problems are such that every column has at most two nonzeros in the constraint matrix. However, the requirement that these coefficients be +1 and -1 is relaxed. Specialized procedures, which may be twenty times faster than the regular simplex method, exist for solving these problems.

Examples of “shipments” with such gains or losses are: investment in an interest-bearing account, electrical transmission with loss, natural gas pipeline shipments where the pipeline pumps burn natural gas from the pipeline, and work force attrition. Stroup and Wollmer (1992) show how a network with gains model is useful in the airline industry for deciding where to purchase fuel and where to ferry fuel from one stop to another. Truemper (1976) points out, if the network with gains has no circuits when considered as an undirected network, then it can be converted to a pure network model by appropriate scaling.

2. *Undirected Networks.* In communications networks, there is typically no direction of shipment. The arcs are undirected.
3. *Multicommodity Networks.* In many distribution situations, there are multiple commodities moving through the network, all competing for scarce network capacity. Each source may produce only one of the commodities and each destination, or sink, may accept only one specific commodity.
4. *Leontief Flow.* In a so-called Leontief input-output model (see Leontief, 1951), each activity uses several commodities although it produces only one commodity. For example, one unit of automotive production may use a half ton of steel, 300 pounds of plastic, and 100 pounds of glass. Material Requirements Planning (MRP) models have the same feature. If each output required only one input, then we would simply have a network with gains. Special purpose algorithms exist for solving Leontief Flow and MRP models. See, for example, Jeroslow, Martin, Rardin, and Wang (1992).
5. *Activity/Resource Diagrams.* If Leontief flow models are extended, so each activity can have not only several inputs, but also several outputs, then one can in fact represent arbitrary LPs. We call the obvious extension of the network diagrams to this case an activity/resource diagram.

8.9.1 Multicommodity Network Flows

In a network LP, one assumption is a customer is indifferent, except perhaps for cost, to the source from which his product was obtained. Another assumption is that there is a single commodity flowing through the network. In many network-like situations, there are multiple distinct commodities flowing through the network. If each link has infinite capacity, then an independent network flow LP could be solved for each commodity. However, if a link has a finite capacity that applies to the sum of all commodities flowing over that link, then we have a multicommodity network problem.

The most common setting for multicommodity network problems is in shipping. The network might be a natural gas pipeline network and the commodities might be different fuels shipped over the network. In other shipping problems, such as traffic assignment or overnight package delivery, each origin/destination pair constitutes a commodity.

The crucial feature is identity of the commodities must be maintained throughout the network. That is, customers care which commodity gets delivered. An example is a metals supply company that ships

aluminum bars, stainless steel rings, steel beams, etc., all around the country, using a single limited capacity fleet of trucks.

In general form, the multicommodity network problem is defined as:

D_{ik} = demand for commodity k at node i , with negative values denoting supply;

C_{ijk} = cost per unit of shipping commodity k from node i to node j ;

U_{ij} = capacity of the link from node i to node j .

We want to find:

X_{ijk} = amount of commodity k shipped from node i to node j , so as to:

$$\min \quad \sum_i \sum_j \sum_k c_{ijk} x_{ijk}$$

subject to:

For each commodity k and node i :

$$\sum_i x_{itk} = D_{tk} + \sum_j x_{tjk}$$

For each link i, j :

$$\sum_k x_{ijk} \leq U_{ij}$$

8.9.2 Reducing the Size of Multicommodity Problems

If the multiple commodities correspond to origin destination pairs and the cost of shipping a unit over a link is independent of the final destination, then you can aggregate commodities over destinations. That is, you need identify a commodity only by its origin, not by both origin and destination. Thus, you have as many commodities as there are origins, rather than (number of origins) \times (number of destinations). For example, in a 100-city problem, using this observation, you would have only 100 commodities, rather than 10,000 commodities.

One of the biggest examples of multicommodity network problems in existence are the Patient Distribution System models developed by the United States Air Force for planning for transport of sick or wounded personnel.

8.9.3 Multicommodity Flow Example

You have decided to compete with Federal Express by offering “point to point” shipment of materials. Starting small, you have identified six cities as the ones you will first serve. The matrix below represents the average number of tons potential customers need to move between each origin/destination pair per day. For example, people in city 2 need to move four tons per day to city 3:

		Demand in tons, $D(i, j)$, by O/D pair						Cost/ton shipped, $C(i, j)$, by link						Capacity in tons, $U(i, j)$, By link						
		1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6	
From	To:	1	0	5	9	7	0	4	0	4	5	8	9	9	0	2	3	2	1	20
		2	0	0	4	0	1	0	3	0	3	2	4	6	0	0	2	8	3	9
3	0	0	0	0	0	0	5	3	0	2	3	5		3	0	0	1	3	9	
4	0	0	0	0	0	0	7	3	3	0	5	6		5	4	6	0	5	9	
5	0	4	0	2	0	8	8	5	3	6	0	3		1	0	2	7	0	9	
6	0	0	0	0	0	0	9	7	4	5	5	0		9	9	9	9	9	0	

Rather than use a hub system as Federal Express does, you will ship the materials over a regular directed network. The cost per ton of shipping from any node i to any other node j is denoted by $C(i, j)$. There is an upper limit on the number of tons shipped per day over any link in the network of $U(i, j)$. This capacity restriction applies to the total amount of all goods shipped over that link, regardless of origin or destination. Note $U(i, j)$ and $C(i, j)$ apply to links in the network, whereas $D(i, j)$ applies to origin/destination pairs. This capacity restriction applies only to the directed flow. That is, $U(i, j)$ need not equal $U(j, i)$. It may be that none of the goods shipped from origin i to destination j moves over link (i, j) . It is important goods maintain their identity as they move through the network. Notice city 6 looks like a hub. It has high capacity to and from all other cities.

In order to get a compact formulation, we note only three cities, 1, 2, and 5, are suppliers. Thus, we need keep track of only three commodities in the network, corresponding to the city of origin for the commodity. Define:

$$X_{ijk} = \text{tons shipped from city } i \text{ to city } j \text{ of commodity } k.$$

The resulting formulation is:

```

MODEL:
! Keywords: multi-commodity, network flow, routing;
! Multi-commodity network flow problem;
SETS:
! The nodes in the network;
NODES/1..6/:;
! The set of nodes that are origins;
COMMO(NODES)/1, 2, 5/:;
EDGES(NODES, NODES): D, C, U, V;
NET(EDGES, COMMO) : X;
ENDSETS
DATA:
! Demand: amount to be shipped from
!          origin(row) to destination(col);
D = 0 5 9 7 0 4
      0 0 4 0 1 0
      0 0 0 0 0 0
      0 0 0 0 0 0
      0 4 0 2 0 8
      0 0 0 0 0 0;
! Cost per unit shipped over a arc/link;
C = 0 4 5 8 9 9
      3 0 3 2 4 6
      5 3 0 2 3 5
      7 3 3 0 5 6
      8 5 3 6 0 3
      9 7 4 5 5 0;
! Upper limit on amount shipped on each link;
U = 0 2 3 2 1 20
      0 0 2 8 3 9
      3 0 0 1 3 9
      5 4 6 0 5 9
      1 0 2 7 0 9
      9 9 9 9 9 0;

```

```

! Whether an arc/link exists or not;
! V = 0 if U = 0;
! V = 1 otherwise;
V = 0 1 1 1 1 1
    0 0 1 1 1 1
    1 0 0 1 1 1
    1 1 1 0 1 1
    1 0 1 1 0 1
    1 1 1 1 1 0;
ENDDATA

! Minimize shipping cost over all links;
MIN = @SUM( NET(I, J, K): C(I, J) * X(I, J, K));

! This is the balance constraint. There are two cases:
! Either the node that needs to be balanced is not a supply,
in which case the sum of incoming amounts
minus the sum of outgoing amounts must equal
the demand for that commodity for that city;
!or where the node is a supply,
the sum of incoming minus outgoing amounts must equal
the negative of the sum of the demand for the commodity
that the node supplies;
@FOR(COMMO(K): @FOR(NODES(J)|J #NE# K:
    @SUM(NODES(I): V(I, J) * X(I, J, K) - V(J, I) * X(J, I, K))
    = D(K, J);
);
@FOR(NODES(J)|J #EQ# K:
    @SUM(NODES(I): V(I, J) * X(I, J, K) - V(J, I) * X(J, I, K))
    = -@SUM( NODES(L): D(K, L)));
);

! This is a capacity constraint;
@FOR(EDGES(I, J)|I #NE# J:
    @SUM(COMMO(K): X(I, J, K)) <= U(I, J);
);
END

```

Notice there are 3 (commodities) \times 6 (cities) = 18 balance constraints. If we instead identified goods by origin/destination combination rather than just origin, there would be $9 \times 6 = 54$ balance constraints. Solving, we get:

Optimal solution found at step:	56	
Objective value:	361.0000	
Variable	Value	Reduced Cost
X(1, 2, 1)	2.000000	0.0000000
X(1, 3, 1)	3.000000	0.0000000
X(1, 4, 1)	2.000000	0.0000000
X(1, 5, 1)	1.000000	0.0000000
X(1, 6, 1)	17.000000	0.0000000
X(2, 3, 2)	2.000000	0.0000000
X(2, 4, 2)	2.000000	0.0000000
X(2, 5, 2)	1.000000	0.0000000
X(3, 4, 5)	1.000000	0.0000000
X(4, 2, 5)	4.000000	0.0000000
X(4, 3, 2)	2.000000	0.0000000

X(5, 3, 1)	1.000000	0.0000000
X(5, 3, 5)	1.000000	0.0000000
X(5, 4, 5)	5.000000	0.0000000
X(5, 6, 5)	8.000000	0.0000000
X(6, 2, 1)	3.000000	0.0000000
X(6, 3, 1)	5.000000	0.0000000
X(6, 4, 1)	5.000000	0.0000000

Notice, because of capacity limitations on other links, the depot city (6) is used for many of the shipments.

8.9.4 Fleet Routing and Assignment

An important problem in the airline and trucking industry is fleet routing and assignment. Given a set of shipments or flights to be made, the routing part is concerned with the path each vehicle takes. This is sometimes called the FTL(Full Truck Load) routing problem. The assignment part is of interest if the firm has several different fleets of vehicles available. Then the question is what type of vehicle is assigned to each flight or shipment. We will describe a simplified version of the approach used by Subramanian et al. (1994) to do fleet assignment at Delta Airlines. A similar approach has been used at US Airways by Kontogiorgis and Acharya (1999).

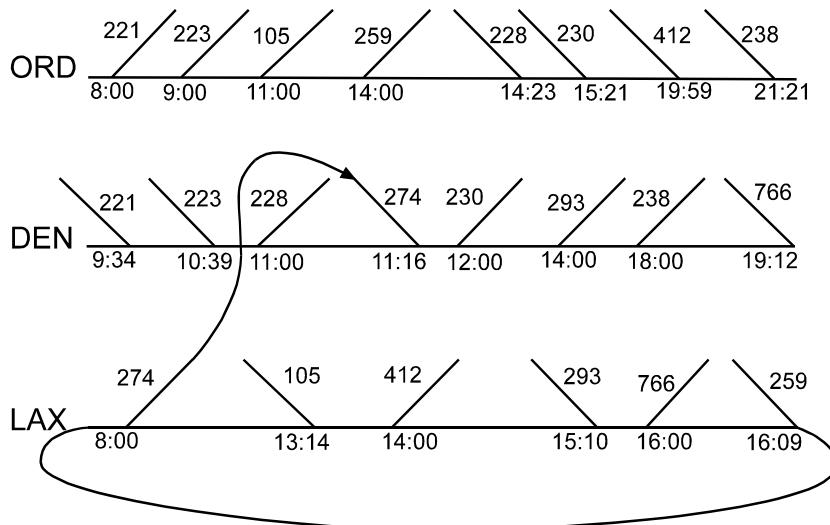
To motivate things, consider the following set of flights serving Chicago (ORD), Denver (DEN), and Los Angeles (LAX) that United Airlines once offered on a typical weekday:

Daily Flight Schedule

Flight	City		Time	
	Depart	Arrive	Depart	Arrive
1	221	ORD	DEN	0800 0934
2	223	ORD	DEN	0900 1039
3	274	LAX	DEN	0800 1116
4	105	ORD	LAX	1100 1314
5	228	DEN	ORD	1100 1423
6	230	DEN	ORD	1200 1521
7	259	ORD	LAX	1400 1609
8	293	DEN	LAX	1400 1510
9	412	LAX	ORD	1400 1959
10	766	LAX	DEN	1600 1912
11	238	DEN	ORD	1800 2121

This schedule can be represented by the network in Figure 8.8. The diagonal lines from upper left to lower right represent flight arrivals. The diagonal lines from lower left to upper right represent departures. To complete the diagram, we need to add the lines connecting each flight departure to each flight arrival. The thin line connecting the departure of Flight 274 from LAX to the arrival of Flight 274 in Denver illustrates one of the missing lines. If the schedule repeats every day, it is reasonable to have the network have a backloop for each city, as illustrated for LAX. To avoid clutter, these lines have not been added.

Figure 8.8 A Fleet Routing Network



Perhaps the obvious way of interpreting this as a network problem is as follows:

- Each diagonal line (with the connection to its partner) constitutes a variable, corresponding to a flight;
- each horizontal line or backloop corresponds to a decision variable representing the number of aircraft on the ground;
- each point of either an arrival or a departure constitutes a node, and the model will have a constraint saying, in words:

$$\begin{aligned}
 & (\text{no. of aircraft on the ground at this city at this instant}) + (\text{arrivals at this instant}) \\
 & = (\text{no. of departures from this city at this instant}) + (\text{no. of aircraft on the ground after this instant}).
 \end{aligned}$$

With this convention, there would be 22 constraints (8 at *ORD*, 8 at *DEN*, and 6 at *LAX*), and 33 variables (11 flight variables and 22 ground variables). The number of constraints and variables can be reduced substantially if we make the observation that the feasibility of a solution is not affected if, for each city:

- Each arrival is delayed until the first departure after that arrival.
- Each departure is advanced (made earlier) to the most recent departure just after an arrival.

Thus, the only nodes required are when a departure immediately follows an arrival.

If we have a fleet of just one type of aircraft, we probably want to know what is the minimum number of aircrafts needed to fly this schedule. In words, our model is:

Minimize number of aircraft on the ground overnight
 (That is the only place they can be, given the flight schedule)
 subject to

source of aircraft = use of aircraft at each node of the network
 and each flight must be covered.

Taking all the above observations into account gives the following formulation of a network LP. Note the G variables represent the number of aircraft on the ground at a given city just after a specified instant:

```

! Fleet routing with a single plane type;
! Minimize number of planes on ground overnight;
MIN = GC2400 + GD2400 + GL2400;
! The plane(old) conservation constraints;
! Chicago at 8 am, sources - uses = 0;
GC2400 - F221 - F223 - F105 - F259 - GC1400 = 0;
! Chicago at midnight;
GC1400 + F228 + F230 + F412 + F238 - GC2400 = 0;
! Denver at 11 am;
GD2400 + F221 + F223 - F228 - GD1100 = 0;
! Denver at high noon;
GD1100 + F274 - F230 - F293 - F238 - GD1800 = 0;
! Denver at midnight;
GD1800 + F766 - GD2400 = 0;
! LA at 8 am;
GL2400 - F274 - GL0800 = 0;
! LA at 1400;
GL0800 + F105 - F412 - GL1400 = 0;
! LA at 1600;
GL1400 + F293 - F766 - GL1600 = 0;
! LA at midnight;
GL1600 + F259 - GL2400 = 0;
! Cover our flight's constraints;
F221 = 1;
F223 = 1;
F274 = 1;
F105 = 1;
F228 = 1;
F230 = 1;
F259 = 1;
F293 = 1;
F412 = 1;
F766 = 1;
F238 = 1;
```

This model assumes no deadheading is used. That is, no plane is flown empty from one city to another in order to position it for the next day. The reader probably figured out by simple intuitive arguments that six aircraft are needed. The following solution gives the details:

Optimal solution found at step:	0	
Objective value:	6.000000	
Variable	Value	Reduced Cost
GC2400	4.000000	0.0000000
GD2400	1.000000	0.0000000
GL2400	1.000000	0.0000000
F221	1.000000	0.0000000
F223	1.000000	0.0000000
F105	1.000000	0.0000000
F259	1.000000	0.0000000
GC1400	0.0000000	1.000000
F228	1.000000	0.0000000
F230	1.000000	0.0000000
F412	1.000000	0.0000000
F238	1.000000	0.0000000
GD1100	2.000000	0.0000000
F274	1.000000	0.0000000
F293	1.000000	0.0000000
GD1800	0.0000000	1.000000
F766	1.000000	0.0000000
GL0800	0.0000000	0.0000000
GL1400	0.0000000	0.0000000
GL1600	0.0000000	1.000000

Thus, there are four aircraft on the ground overnight at Chicago, one overnight at Denver, and one overnight at Los Angeles.

8.9.5 Fleet Assignment

If we have two or more aircraft types, then we have the additional decision of specifying the type of aircraft assigned to each flight. The typical setting is we have a limited number of new aircraft that are more efficient than previous aircraft. Let us extend our previous example by assuming we have two aircraft of type B. They are more fuel-efficient than our original type A aircraft. However, the capacity of type B is slightly less than A. We now probably want to maximize the profit contribution. The profit contribution from assigning an aircraft of type i to flight j is:

- + (revenue from satisfying all demand on flight j)
- (“spill” cost of not being able to serve all demand on j because of the limited capacity of aircraft type i)
- (the operating cost of flying aircraft type i on flight j)
- + (revenue from demand spilled from previous flights captured on this flight).

The spill costs and recoveries are probably the most difficult to estimate.

The previous model easily generalizes with the two modifications:

- a) Conservation of flow constraints is needed for each aircraft type.
- b) The flight coverage constraints become more flexible, because there are now two ways of covering a flight.

After carefully calculating the profit contribution for each combination of aircraft type and flight, we get the following model:

```

! Fleet routing and assignment with two plane types;
! Maximize profit contribution from flights covered;
MAX = 105 * F221A + 121 * F221B + 109 * F223A + 108
      * F223B + 110 * F274A + 115 * F274B + 130 *
      F105A + 140 * F105B + 106 * F228A + 122 *
      F228B + 112 * F230A + 115 * F230B + 132 *
      F259A + 129 * F259B + 115 * F293A + 123 *
      F293B + 133 * F412A + 135 * F412B + 108 *
      F766A + 117 * F766B + 116 * F238A + 124 *
      F238B;

! Conservation of flow constraints;
! for type A aircraft;
! Chicago at 8 am, sources - uses = 0;
F221A - F223A - F105A - F259A - GC1400A + GC2400A=0;
! Chicago at midnight;
F228A + F230A + F412A + F238A + GC1400A - GC2400A=0;
! Denver at 11 am;
F221A + F223A - F228A - GD1100A + GD2400A = 0;
! Denver at high noon;
F274A - F230A - F293A - F238A + GD1100A - GD1800A=0;
! Denver at midnight;
F766A - GD2400A + GD1800A = 0;
! LA at 8 am;
- F274A - GL0800A + GL2400A = 0;
! LA at 1400;
F105A - F412A + GL0800A - GL1400A = 0;
! LA at 1600;
F293A - F766A + GL1400A - GL1600A = 0;
! LA at midnight;
F259A - GL2400A + GL1600A = 0;
! Aircraft type B, conservation of flow;
! Chicago at 8 am;
-F221B - F223B - F105B - F259B - GC1400B +GC2400B=0;
! Chicago at midnight;
F228B + F230B + F412B + F238B + GC1400B - GC2400B=0;
! Denver at 11 am;
F221B + F223B - F228B - GD1100B + GD2400B = 0;
! Denver at high noon;
F274B - F230B - F293B - F238B + GD1100B - GD1800B=0;
! Denver at midnight;
F766B - GD2400B + GD1800B = 0;
! LA at 8 am;
- F274B - GL0800B + GL2400B = 0;
! LA at 1400;
F105B - F412B + GL0800B - GL1400B = 0;
! LA at 1600;
F293B - F766B + GL1400B - GL1600B = 0;
! LA at midnight;
F259B - GL2400B + GL1600B = 0;
! Can put at most one plane on each flight;

```

```

F221A + F221B <= 1;
F223A + F223B <= 1;
F274A + F274B <= 1;
F105A + F105B <= 1;
F228A + F228B <= 1;
F230A + F230B <= 1;
F259A + F259B <= 1;
F293A + F293B <= 1;
F412A + F412B <= 1;
F766A + F766B <= 1;
F238A + F238B <= 1;
! Fleet size of type B;
GC2400B + GD2400B + GL2400B <= 2;

```

The not so obvious solution is:

Optimal solution found at step:	37	
Objective value:	1325.000	
Variable	Value	Reduced Cost
F221B	1.000000	0.0000000
F223A	1.000000	0.0000000
F274A	1.000000	0.0000000
F105A	1.000000	0.0000000
F228B	1.000000	0.0000000
F230A	1.000000	0.0000000
F259A	1.000000	0.0000000
F293B	1.000000	0.0000000
F412A	1.000000	0.0000000
F766B	1.000000	0.0000000
F238A	1.000000	0.0000000
GC2400A	3.000000	0.0000000
GD1100A	1.000000	0.0000000
GL2400A	1.000000	0.0000000
GC2400B	1.000000	0.0000000
GD1100B	1.000000	0.0000000
GD2400B	1.000000	0.0000000

Six aircraft are still used. The newer type *B* aircraft cover flights 221, 228, 293, and 766. Since there are two vehicle types, this model is a multicommodity network flow model rather than a pure network flow model. Thus, we are not guaranteed to be able to find a naturally integer optimal solution to the LP. Nevertheless, such was the case for the example above.

Generating an explicit model as above would be tedious. The following is a set-based version of the above model. With the set based version, adding a flight or an aircraft type is a fairly simple clerical operation:

```

MODEL:
SETS: ! Fleet routing and assignment (FLEETRAV);
CITY :; ! The cities involved;
ACRFT: ! Aircraft types;
FCOST, ! Fixed cost per day of this type;
FSIZE; ! Max fleet size of this type;
FLIGHT:;
FXCXC( FLIGHT, CITY, CITY) :
  DEPAT, ! Flight departure time;
  ARVAT; ! arrival time at dest. ;
AXC( ACRFT, CITY):
  OVNITE; ! Number staying overnite by type,city;
AXF( ACRFT, FXCXC):
  X, ! Number aircraft used by type,flight;
  PC; ! Profit contribution by type,flight;
ENDSETS
DATA:
CITY = ORD DEN LAX;
ACRFT, FCOST, FSIZE =
  MD90      0      7
  B737      0      2;
FLIGHT = F221 F223 F274 F105 F228 F230 F259 F293 F412 F766 F238;
FXCXC, DEPAT, ARVAT =
!     Flight Origin Dest. Depart Arrive;
  F221    ORD    DEN    800    934
  F223    ORD    DEN    900   1039
  F274    LAX    DEN    800   1116
  F105    ORD    LAX   1100   1314
  F228    DEN    ORD   1100   1423
  F230    DEN    ORD   1200   1521
  F259    ORD    LAX   1400   1609
  F293    DEN    LAX   1400   1510
  F412    LAX    ORD   1400   1959
  F766    LAX    DEN   1600   1912
  F238    DEN    ORD   1800   2121;
PC = ! Profit contribution of each vehicle*flight combo;
  105      109      110      130      106      112
  132      115      133      108      116
  121      108      115      140      122      115
  129      123      135      117      124;
ENDDATA
!-----;
! Maximize profit contribution from flights minus
  overhead cost of aircraft in fleet;
MAX = @SUM( AXF( I, N, J, K): PC( I, N, J, K) * X( I, N, J, K))
  - @SUM( AXC( I, J): FCOST( I) * OVNITE( I, J));

```

```

! At any instant, departures in particular, the number of
cumulative arrivals must be >= number of cumulative departures;
! For each flight of each aircraft type;
@FOR( ACRFT( I):
    @FOR( FXCXC( N, J, K):
        ! Aircraft on ground in morning +
            number aircraft arrived thus far >=
            number aircraft departed thus far;
            OVNITE( I, J) +
                @SUM( FXCXC( N1, J1, K1)| K1 #EQ# J #AND#
                    ARVAT( N1, J1, K1) #LT# DEPAT( N, J, K):
                    X( I, N1, J1, J)) >=
                @SUM( FXCXC( N1, J1, K1)| J1 #EQ# J #AND#
                    DEPAT( N1, J1, K1) #LE# DEPAT( N, J, K):
                    X( I, N1, J, K1));
            );
        );
    ! This model does not allow deadheading, so at the end of the day,
        arrivals must equal departures;
    @FOR( ACRFT( I):
        @FOR( CITY( J):
            @SUM( AXF( I, N, J1, J): X( I, N, J1, J)) =
            @SUM( AXF( I, N, J, K): X( I, N, J, K));
        );
    );
    ! Each flight must be covered;
    @FOR( FXCXC( N, J, K):
        @SUM( AXF( I, N, J, K): X( I, N, J, K)) = 1;
    );
    ! Fleet size limits;
    @FOR( ACRFT( I):
        @SUM( AXC( I, J): OVNITE( I, J)) <= FSIZE( I);
    );
    ! Fractional planes are not allowed;
    @FOR( AXF: @GIN( X); );
END

```

Sometimes, especially in trucking, one has the option of using rented vehicles to cover only selected trips. With regard to the model, the major modification is that rented vehicles do not have to honor the conservation of flow constraints. Other details that are sometimes included relate to maintenance. With aircraft, for example, a specific aircraft must be taken out of service for maintenance after a specified number of landings, or after a specified number of flying hours, or after a certain elapsed time, whichever occurs first. It is not too difficult to incorporate such details, although the model becomes substantially larger.

8.9.6 Leontief Flow Models

In a Leontief flow model, each activity produces one output. However, it may use zero or more inputs. The following example illustrates.

Example: Islandia Input-Output Model

The country of Islandia has four major export industries: steel, automotive, electronics, and plastics. The economic minister of Islandia would like to maximize exports-imports. The unit of exchange in Islandia is the klutz. The prices in klutzes on the world market per unit of steel, automotive,

electronics, and plastics are, respectively: 500, 1500, 300, and 1200. Production of one unit of steel requires 0.02 units of automotive production, 0.01 units of plastics, 250 klutzes of raw material purchased on the world market, plus one-half man-year of labor. Production of one automotive unit requires 0.8 units of steel, 0.15 units of electronics, 0.11 units of plastic, one man-year of labor, and 300 klutzes of imported material. Production of one unit of electronic equipment requires 0.01 units of steel, 0.01 units of automotive, 0.05 units of plastic, half a man-year of labor, and 50 klutzes of imported material. Automotive production is limited at 650,000 units. Production of one unit of plastic requires 0.03 units of automotive production, 0.2 units of steel, 0.05 units of electronics, 2 man-years of labor, plus 300 klutzes of imported materials. The upper limit on plastic is 60,000 units. The total manpower available in Islandia is 830,000 men per year. No steel, automotive, electronics, or plastic products may be imported.

How much should be produced and exported of the various products?

Formulation and Solution of the Islandia Problem

The formulation of an input-output model should follow the same two-step procedure for formulating any LP model. Namely, (1) identify the decision variables and (2) identify the constraints. The key to identifying the decision variables for this problem is to make the distinction between the amount of commodity produced and the amount exported. Once this is done, the decision variables can be represented as:

PROD(STEEL)	=	units of steel produced,
PROD(AUTO)	=	units of automotive produced,
PROD(PLASTIC)	=	units of plastic produced,
PROD(ELECT)	=	units of electronics produced,
EXP(STEEL)	=	units of steel exported,
EXP(AUTO)	=	units of automotive exported,
EXP(PLASTIC)	=	units of plastic exported,
EXP(ELECT)	=	units of electronics exported.

The commodities can be straightforwardly identified as steel, automotive, electronics, plastics, manpower, automotive capacity, and plastics capacity. Thus, there will be seven constraints.

The sets formulation and solution are:

```

MODEL: ! Islandia Input/output model;
SETS:
  COMMO:
    PROD, EXP, REV, COST, MANLAB, CAP;
    CXC(COMMO, COMMO): USERATE;
  ENDSETS
  DATA:
    COMMO = STEEL, AUTO, PLASTIC, ELECT;
    COST = 250   300   300   50;
    REV = 500   1500  1200  300;
    MANLAB = .5   1     2     .5;
    ! Amount used of the column commodity per unit
    ! of the row commodity;
    USERATE= -1   .02   .01   0
              .8   -1   .11   .15
              .2   .03   -1   .05
              .01  .01   .05   -1;
    MANPOWER = 830000;
    CAP = 999999 650000 60000 999999;
  ENDDATA
  [PROFIT] MAX = @SUM( COMMO: REV * EXP - PROD * COST);
  @FOR( COMMO( I):
    [NETUSE] ! Net use must equal = 0;
    EXP(I) + @SUM(COMMO(J): USERATE(J,I)* PROD(J)) = 0;
    [CAPLIM] PROD( I) <= CAP( I);
  );
  [MANLIM] @SUM(COMMO:PROD * MANLAB) < MANPOWER;
END

```

Notice this model has the Leontief flow feature. Namely, each decision variable has at most one negative constraint coefficient.

The solution is:

```

Global optimal solution found.
Objective value: 0.4354312E+09

```

Variable	Value	Reduced Cost
MANPOWER	830000.0	0.000000
PROD(STEEL)	393958.3	0.000000
PROD(AUTO)	475833.3	0.000000
PROD(PLASTIC)	60000.00	0.000000
PROD(ELECT)	74375.00	0.000000
EXP(STEEL)	547.9167	0.000000
EXP(AUTO)	465410.4	0.000000
EXP(PLASTIC)	0.000000	2096.875
EXP(ELECT)	0.000000	121.8750
REV(STEEL)	500.0000	0.000000
REV(AUTO)	1500.0000	0.000000
REV(PLASTIC)	1200.0000	0.000000

REV(ELECT)	300.0000	0.000000
COST(STEEL)	250.0000	0.000000
COST(AUTO)	300.0000	0.000000
COST(PLASTIC)	300.0000	0.000000
COST(ELECT)	50.00000	0.000000
MANLAB(STEEL)	0.5000000	0.000000
MANLAB(AUTO)	1.0000000	0.000000
MANLAB(PLASTIC)	2.0000000	0.000000
MANLAB(ELECT)	0.5000000	0.000000
CAP(STEEL)	999999.0	0.000000
CAP(AUTO)	650000.0	0.000000
CAP(PLASTIC)	60000.00	0.000000
CAP(ELECT)	999999.0	0.000000
USERATE(STEEL, STEEL)	-1.000000	0.000000
USERATE(STEEL, AUTO)	0.2000000E-01	0.000000
USERATE(STEEL, PLASTIC)	0.1000000E-01	0.000000
USERATE(STEEL, ELECT)	0.000000	0.000000
USERATE(AUTO, STEEL)	0.8000000	0.000000
USERATE(AUTO, AUTO)	-1.000000	0.000000
USERATE(AUTO, PLASTIC)	0.1100000	0.000000
USERATE(AUTO, ELECT)	0.1500000	0.000000
USERATE(PLASTIC, STEEL)	0.2000000	0.000000
USERATE(PLASTIC, AUTO)	0.3000000E-01	0.000000
USERATE(PLASTIC, PLASTIC)	-1.000000	0.000000
USERATE(PLASTIC, ELECT)	0.5000000E-01	0.000000
USERATE(ELECT, STEEL)	0.1000000E-01	0.000000
USERATE(ELECT, AUTO)	0.1000000E-01	0.000000
USERATE(ELECT, PLASTIC)	0.5000000E-01	0.000000
USERATE(ELECT, ELECT)	-1.000000	0.000000

Row	Slack or Surplus	Dual Price
PROFIT	0.4354312E+09	1.000000
NETUSE(STEEL)	0.000000	500.0000
CAPLIM(STEEL)	606040.7	0.000000
NETUSE(AUTO)	0.000000	1500.000
CAPLIM(AUTO)	174166.7	0.000000
NETUSE(PLASTIC)	0.000000	3296.875
CAPLIM(PLASTIC)	0.000000	2082.656
NETUSE(ELECT)	0.000000	421.8750
CAPLIM(ELECT)	925624.0	0.000000
MANLIM	0.000000	374.0625

The solution indicates the best way of selling Islandia's steel, automotive, electronics, plastics, and manpower resources is in the form of automobiles.

This problem would fit the classical input-output model format of Leontief if, instead of maximizing profits, target levels were set for the export (or consumption) of steel, automotive, and plastics. The problem would then be to determine the production levels necessary to support the specified export/consumption levels. In this case, the objective function is irrelevant.

A natural generalization is to allow alternative technologies for producing various commodities. These various technologies may correspond to the degree of mechanization or the form of energy consumed (e.g., gas, coal, or hydroelectric).

8.9.7 Activity/Resource Diagrams

The graphical approach for depicting a model can be extended to arbitrary LP models. The price one must pay to represent a general LP graphically is one must introduce an additional component type into the network. There are two component types in such a diagram: (1) activities, which correspond to variables and are denoted by a square, and (2) resources, which correspond to constraints and are denoted by a circle. Each constraint can be thought of as corresponding to some commodity and, in words, as saying “uses of commodity \leq sources of commodity”. The arrows incident to a square correspond to the resources, commodities, or constraints with which that variable has an interaction. The arrows incident to a circle must obviously then correspond to the activities or decision variables with which the constraint has an interaction.

Example: The Vertically Integrated Farmer

A farmer has 120 acres that can be used for growing wheat or corn. The yield is 55 bushels of wheat or 95 bushels of corn per acre per year. Any fraction of the 120 acres can be devoted to growing wheat or corn. Labor requirements are 4 hours per acre per year, plus 0.15 hours per bushel of wheat, and 0.70 hours per bushel of corn. Cost of seed, fertilizer, etc., is 20 cents per bushel of wheat produced and 12 cents per bushel of corn produced. Wheat can be sold for \$1.75 per bushel and corn for \$0.95 per bushel. Wheat can be bought for \$2.50 per bushel and corn for \$1.50 per bushel.

In addition, the farmer may raise pigs and/or poultry. The farmer sells the pigs or poultry when they reach the age of one year. A pig sells for \$40. He measures the poultry in terms of coops. One coop brings in \$40 at the time of sale. One pig requires 25 bushels of wheat or 20 bushels of corn, plus 25 hours of labor and 25 square feet of floor space. One coop of poultry requires 25 bushels of corn or 10 bushels of wheat, plus 40 hours of labor and 15 square feet of floor space.

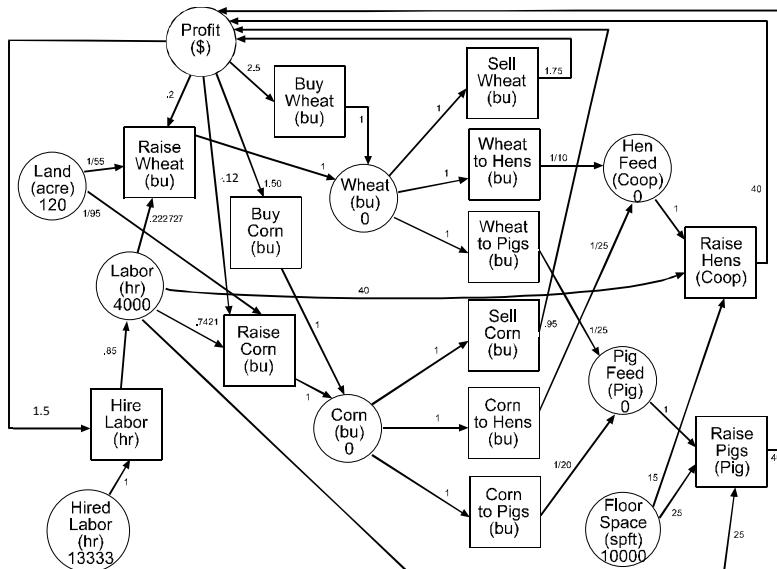
The farmer has 10,000 square feet of floor space. He has available per year 2,000 hours of his own time and another 2,000 hours from his family. He can hire labor at \$1.50 per hour. However, for each hour of hired labor, 0.15 hour of the farmer's time is required for supervision. How much land should be devoted to corn and to wheat, and how many pigs and/or poultry should be raised to maximize the farmer's profits? This problem is based on an example in chapter 12 of Hadley (1962).

You may find it convenient to use the following variables for this problem:

WR	Wheat raised (in bushels)
CR	Corn harvested (in bushels)
PS	Pigs raised and sold
HS	Hens raised and sold (number of coops)
LB	Labor hired (in hours)
WS	Wheat marketed or sold (in bushels)
CS	Corn marketed or sold (in bushels)
CH	Corn used to feed hens (in bushels)
WH	Wheat used to feed hens (in bushels)
CP	Corn used to feed pigs (in bushels)
WP	Wheat used to feed pigs (in bushels)
CB	Corn bought (in bushels)
WB	Wheat bought (in bushels)

The activity-resource diagram for the preceding problem is shown in Figure 8.9:

Figure 8.9 An Activity-Resource Diagram



Some things to note about an activity-resource diagram are:

- Each rectangle in the diagram corresponds to a decision variable in the formulation.
- Each circle in the diagram corresponds to a constraint or the objective.
- Each arrow in the diagram corresponds to a coefficient in the formulation.
- Associated with each circle or rectangle is a unit of measure (e.g., hours or bushels).
- The units or dimension of each arrow is:
“Units of the circle” per “unit of the rectangle.”

Below is the formulation corresponding to the above diagram.

```

! All constraints are in Uses <= Sources form;
[PROFIT] MAX= 1.75*WS + .95*CS +40*PS +40*HS -1.5*LB -.2*WR -.12*CR -1.5*CB -2.5*WB;
[LAND] (1/55)*WR + (1/95)*CR <= 120 ; ! Acres;
[LABOR] (.15+4/55)*WR + (.7+4/95)*CR + 40*HS + 25*PS <= .85*LB + 4000 ; ! Hours;
[HRDLABOR] LB <= 2000/.15; ! Hours;
[WHEAT] WS + WH + WP <= WB + WR; ! Bushels);
[CORN] CS + CH + CP <= CB + CR; ! Bushels;
[HENFEED] HS <= (1/25)*CH + (1/10)*WH; ! Coops;
[PIGFEED] PS <= (1/20)*CP + (1/25)*WP; ! Pigs;
[FLOORSP] 25*PS + 15*HS <= 10000; ! Square feet;

```

Notice that there is a one-to-one correspondence between the rows of the formulation and the round nodes of the diagram, and a one-to-one correspondence between the variables of the formulation and the square “hyper-arcs” of the diagram. The solution is:

Variable	Value	Reduced Cost
WS	5967.500	0.000000
CS	0.000000	0.4122697
PS	0.000000	0.000000
HS	63.25000	0.000000
LB	0.000000	1.021875
WR	6600.000	0.000000
CR	0.000000	0.000000
CB	0.000000	0.1377303
WB	0.000000	0.7500000
WH	632.5000	0.000000
WP	0.000000	0.7125000
CH	0.000000	0.6622697
CP	0.000000	0.0653947

Row	Slack or Surplus	Dual Price
PROFIT	11653.12	1.000000
LAND	0.000000	78.35938
LABOR	0.000000	0.5625000
HRDLABOR	13333.33	0.000000
WHEAT	0.000000	1.750000
CORN	0.000000	1.362270
HENFEED	0.000000	17.50000
PIGFEED	0.000000	25.93750
FLOORSP	9051.250	0.000000

Notice that the most profitable use of land is to raise wheat. The most profitable use of the farmer’s own labor and floor space is to use it, plus some wheat, to raise hens.

8.9.8 Spanning Trees

Another simple yet important network-related problem is the spanning tree problem. It arises, for example, in the installation of utilities such as cable, power lines, roads, and sewers to provide services to homes in newly developed regions. Given a set of homes to be connected, we want to find a minimum cost network, so every home is connected to the network. A reasonable approximation to the cost of the network is the sum of the costs of the arcs in the network. If the arcs have positive costs, then a little reflection should convince you the minimum cost network contains no loops (i.e., for any

two nodes (or homes) on the network, there is exactly one path connecting them). Such a network is called a spanning tree.

A simple algorithm is available for finding a minimal cost spanning tree, see Kruskal (1956):

1. Set $Y = \{2, 3, 4 \dots n\}$ (i.e., the set of nodes *yet to be connected*).
 $A = \{1\}$ (i.e., the set of *already connected nodes*). We may arbitrarily define node 1 as the root of the tree.
2. If Y is empty, then we are done,
3. else find the shortest arc (i,j) such that i is in A and j is in Y .
4. Add arc (i,j) to the network and
set $A = A + j$,
 $Y = Y - j$.
5. Go to (2).

Because of the above simple and efficient algorithm, LP is not needed to solve the minimum spanning tree problem. In fact, formulating the minimum spanning tree problem as an LP is a bit tedious.

The following illustrates a LINGO model for a spanning tree. This model does not explicitly solve it as above, but just solves it as a straightforward integer program:

```

MODEL:      ! (MNSPTREE);
!Given a set of nodes and the distance between each pair, find
the shortest total distance of links on the network to connect
all the nodes. This is the classic minimal spanning tree (MST)
problem;
SETS:
CITY: LVL;
! LVL( I) = level of city I in tree. LVL( 1) = 0;
LINK( CITY, CITY):
DIST,   ! The distance matrix;
X;      ! X( I,J) = 1 if we use link I, J;
ENDSETS
! This model finds the minimum cost network connecting Atlanta,
Chicago, Cincinnati, Houston, LA, and Montreal so that
messages can be sent from Atlanta (base) to all other cities;
DATA:
CITY= ATL  CHI  CIN  HOU  LAX  MON;
! Distance matrix need not be symmetric. City 1 is base;
DIST =  0  702  454  842  2396 1196 !from Atlanta;
        702  0  324 1093 2136  764 !from Chicago;
        454  324  0 1137 2180  798 !from Cinci;
        842 1093 1137  0 1616 1857 !from Houston;
        2396 2136 2180 1616  0 2900 !from LA;
        1196  764  798 1857 2900  0;!from Montreal;
ENDDATA
!-----
!The model size: Warning, may be slow for N > 8;
N = @SIZE( CITY);
!The objective is to minimize total dist. of links;
MIN = @SUM( LINK: DIST * X);

```

```

!For city K, except the base, ... ;
@FOR( CITY( K) | K #GT# 1: ! It must be entered;
      @SUM( CITY( I) | I #NE# K: X( I, K)) = 1;
!If there is a link from J-K, then LVL(K)=LVL(J)+1.
Note:These are not very powerful for large problems;
@FOR( CITY( J) | J #NE# K:
      LVL( K) >= LVL( J) + X( J, K)
      - ( N - 2) * ( 1 - X( J, K))
      + ( N - 3) * X( K, J); );
LVL( 1) = 0; ! City 1 has level 0;
!There must be an arc out of city 1;
@SUM( CITY( J) | J #GT# 1: X( 1, J)) >= 1;
!Make the X's 0/1;
@FOR( LINK: @BIN( X); );
!The level of a city except the base is at least 1 but no more than
N-1, and is 1 if link to the base;
@FOR( CITY( K) | K #GT# 1:
      @BND( 1, LVL( K), 999999);
      LVL( K) <= N - 1 - ( N - 2) * X( 1, K); );
END

```

The solution is:

Optimal solution found at step:	16	
Objective value:	4000.000	
Variable		Reduced Cost
N		0.0000000
LVL(CHI)		0.0000000
LVL(CIN)		0.0000000
LVL(HOU)		0.0000000
LVL(LAX)		0.0000000
LVL(MON)		0.0000000
X(ATL, CIN)	1.000000	454.0000
X(ATL, HOU)	1.000000	842.0000
X(CHI, MON)	1.000000	764.0000
X(CIN, CHI)	1.000000	324.0000
X(HOU, LAX)	1.000000	1616.000

The solution indicates Atlanta should connect to Cincinnati and Houston. Houston, in turn connects to LA. Cincinnati connects to Chicago, and Chicago connects to Montreal.

8.9.9 Steiner Trees

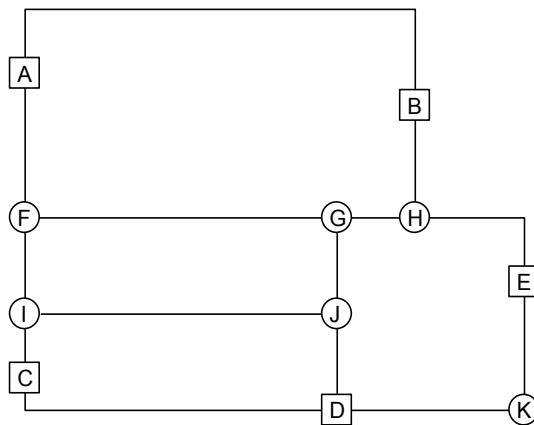
A Steiner tree is a generalization of a minimal spanning tree. The difference is, for a given network, only a specified subset of the nodes need be connected in a Steiner tree. Providing network services in a new housing development is a simple example, such as communication cable, sewer lines, water lines, and roads. Each house must be connected to the network, but not all possible nodes or junctions in the candidate network need be included.

Example

The first computer, an IBM RS6000, to beat a grandmaster, Gary Kasparov, at the game of chess, contained electronic chips designed with the help of Steiner-tree-like optimization methods. A typical VLSI (Very Large Scale Integrated) chip on this computer was less than 2 millimeters on a side.

Nevertheless, it might contain over 120 meters of connecting pathways for connecting the various devices on the chip. An important part of increasing the speed of a chip is reducing the length of the paths on the chip. Figure 8.10 shows a chip on which five devices must be connected on a common tree. Because of previous placement of various devices, the only available links are the ones indicated in the figure. The square nodes, A, B, C, D, and E, must be connected. The round nodes, F, G, H, I, J, and K, may be used, but need not be connected. What set of links should be used to minimize the total distance of links?

Figure 8.10 Steiner Tree Problem



Finding a minimal length Steiner tree is considerably harder than finding a minimal length spanning tree. For small problems, the following LINGO model will find minimal length Steiner trees. The data correspond to the network in Figure 8.10:

```

MODEL:      ! (STEINERT);
!Given a set of nodes, the distance between them, and a specified
subset of the nodes, find the set of links so that the total distance
is minimized, and there is a (unique) path between every pair of
nodes in the specified subset. This is called a Steiner tree problem;
SETS:
ALLNODE : U;
    ! U( I) = level of node I in the tree;
    ! U( 1) = 0;
MUSTNOD( ALLNODE); ! The subset of nodes that must be connected;
LINK( ALLNODE, ALLNODE):
    DIST, ! The distance matrix;
    X;    ! X( I, J) = 1 if we use link I, J;
ENDSETS

```

```

DATA:
  ALLNODE= ! Distance matrix need not be symmetric;
    A   B   C   D   E   F   G   H   I   J   K;
  DIST =0 14 999 999 999 4 999 999 999 999 999
         14 0 999 999 999 999 999 999 3 999 999 999
         999 999 0 9 999 999 999 999 999 2 999 999
         999 999 9 0 999 999 999 999 999 3 6
         999 999 999 999 0 999 999 5 999 999 3
         4 999 999 999 999 0 999 999 3 999 999
         999 999 999 999 999 999 0 2 999 3 999
         999 3 999 999 5 999 2 0 999 999 999
         999 999 2 999 999 3 999 999 0 8 999
         999 999 999 3 999 999 3 999 8 0 999
         999 999 999 6 3 999 999 999 999 999 0;
  ! The subset of nodes that must be connected.
  ! The first node must be a must-do node;
  MUSTNOD = A B C D E;
ENDDATA
!-----
! The model size: Warning, may be slow for N > 8;
N = @SIZE( ALLNODE);
! Objective is minimize total distance of links;
MIN = @SUM( LINK: DIST * X);
! For each must-do node K, except the base, ... ;
@FOR( MUSTNOD( K)| K #GT# 1:
! It must be entered;
  @SUM( ALLNODE( I)| I #NE# K: X( I, K)) = 1;
! Force U(J)=number arcs between node J and node 1. Note: This is not
very strong for large problems;
@FOR( ALLNODE( J)| J #GT# 1 #AND# J #NE# K:
  U( J) >= U( K) + X( K, J) -
    ( N - 2) * ( 1 - X( K, J)) +
    ( N - 3) * X( J, K); );
!
! There must be an arc out of node 1;
@SUM( ALLNODE( J)| J #GT# 1: X( 1, J)) >= 1;
! If an arc out of node J, there must be an arc in;
@FOR( ALLNODE( J)| J #GT# 1:
  @FOR( ALLNODE( K)| K #NE# J:
    @SUM( ALLNODE( I)| I #NE# K #AND# I #NE# J:
      X( I, J)) >= X( J, K);
    ); );
!
! Make the X's 0/1;
@FOR( LINK: @BIN( X); );
!
! Level of a node except the base is at least 1, no more than N-1,
and is 1 if link to the base;
@FOR( ALLNODE( K)| K #GT# 1:
  @BND( 1, U( K), 999999);
  U( K) < N - 1 - ( N - 2) * X( 1, K); );
END

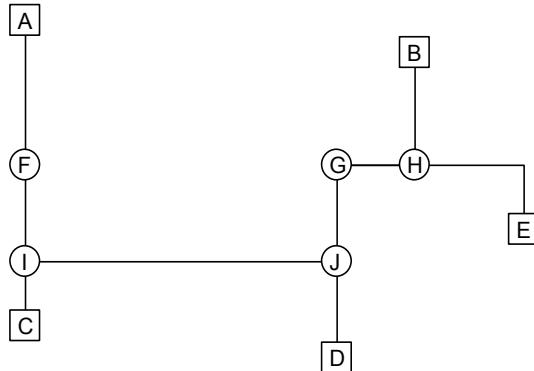
```

The solution has a cost of 33. The list of links used is:

Optimal solution found at step:	30	
Objective value:	33.00000	
Branch count:	0	
Variable	Value	Reduced Cost
X(A, F)	1.000000	4.000000
X(F, I)	1.000000	3.000000
X(G, H)	1.000000	2.000000
X(H, B)	1.000000	3.000000
X(H, E)	1.000000	5.000000
X(I, C)	1.000000	2.000000
X(I, J)	1.000000	8.000000
X(J, D)	1.000000	3.000000
X(J, G)	1.000000	3.000000

The corresponding minimal length Steiner tree appears in Figure 8.11:

Figure 8.11 Minimal Length Steiner Tree



Notice node K is not in the tree. This example is missing two important features of real chip design: a) It shows only two-dimensional paths. In fact, three-dimensional paths are possible (and typically needed) by adding vertical layers to the chip. b) It only shows one tree; whereas, in fact, there may be many distinct trees to be constructed (e.g., some devices need to be connected to electrical “ground”, others need to be connected to the clock signal, etc.). This might be handled by solving the trees sequentially, the more complicated trees first.

8.10 Nonlinear Networks

There are a number of important problems where the constraints describe a network problem. However, either the objective is nonlinear, or there are additional conditions on the network that are nonlinear. The first example describes a transportation problem where the value of shipping or assigning an item to a destination depends upon (a) how many items have already been shipped to that destination, and (b) the type of item and type of destination. In the military, this kind of problem is known as a weapons or target assignment problem. If we define:

$$\begin{aligned}x(i,j) &= \text{number of units of type } j \text{ assigned to task } i; \\ p(i,j) &= \text{Prob}\{\text{a unit of type } j \text{ will not successfully complete task } i\}.\end{aligned}$$

Then, assuming independence, the probability task i will not be completed is:

$$p(i, 1)^{x(i, 1)} p(i, 2)^{x(i, 2)} \dots p(i, n)^{x(i, n)}.$$

The log of the probability that task i will not be completed is:

$$x(i, 1) * \log[p(i, 1)] + x(i, 2) * \log[p(i, 2)] + \dots + x(i, n) * \log[p(i, n)].$$

A reasonable objective is to maximize the expected value of successfully completed tasks. The following model illustrates this idea, using data from Bracken and McCormick (1968):

```

MODEL:
!      (TARGET) Bracken and McCormick;
SETS:
DESTN/1..20/: VALUE, DEM, LFAILP;
SOURCE/1..5/: AVAIL;
```

```

DXS( DESTN, SOURCE): PROB, VOL;
ENDSETS
DATA:
! Probability that a unit from source J will NOT do the job at
destination I;
PROB=
1.00      .84      .96      1.00      .92
.95      .83      .95      1.00      .94
1.00      .85      .96      1.00      .92
1.00      .84      .96      1.00      .95
1.00      .85      .96      1.00      .95
.85      .81      .90      1.00      .98
.90      .81      .92      1.00      .98
.85      .82      .91      1.00      1.00
.80      .80      .92      1.00      1.00
1.00      .86      .95      .96      .90
1.00      1.00      .99      .91      .95
1.00      .98      .98      .92      .96
1.00      1.00      .99      .91      .91
1.00      .88      .98      .92      .98
1.00      .87      .97      .98      .99
1.00      .88      .98      .93      .99
1.00      .85      .95      1.00      1.00
.95      .84      .92      1.00      1.00
1.00      .85      .93      1.00      1.00
1.00      .85      .92      1.00      1.00;
! Units available at each source;
AVAIL= 200      100      300      150      250;
! Min units required at each destination;
DEM=
30      0      0      0      0 100      0      0      0      40
0      0      0      50      70      35      0      0      0      10;
! Value of satisfying destination J;
VALUE=
60      50      50      75      40      60      35      30      25 150
30      45      125      200      200      130      100      100      100 150;
ENDDATA
!Max sum over I:(value of destn I)
*Prob{success at I};
MAX = @SUM( DESTN( I): VALUE( I) *
( 1 - @EXP( LFAILP( I)) ));
! The supply constraints;@FOR( SOURCE( J):
@SUM( DESTN( I): VOL( I, J)) <= AVAIL( J));
@FOR( DESTN( I):
!The demand constraints;
@SUM( SOURCE( J): VOL( I, J)) > DEM( I);
!Compute log of destination I failure probability;
@FREE( LFAILP( I));
LFAILP( I) =
@SUM(SOURCE(J): @LOG(PROB(I,J)) * VOL(I,J)););
END

```

Observe the model could be “simplified” slightly by using the equation computing $LFAILP(I)$ to substitute out $LFAILP(I)$ in the objective. The time required to solve the model would probably increase substantially, however, if this substitution were made. The reason is the number of variables appearing nonlinearly in the objective increases dramatically. A general rule for nonlinear programs is:

If you can reduce the number of variables that appear nonlinearly in the objective or a constraint by using *linear* constraints to define intermediate variables, then it is probably worth doing.

Verify the constraint defining $LFAILP(I)$ is linear in both $LFAILP(I)$ and $VOL(I, J)$.

Notice the solution involves fractional assignments. A simple generalization of the model is to require the $VOL()$ variables to be general integers:

Optimal solution found at step:	152	
Objective value:	1735.570	
Variable	Value	Reduced Cost
$VOL(1, 5)$	50.81594	0.0000000
$VOL(2, 1)$	13.51739	0.0000000
$VOL(2, 2)$	1.360311	0.2176940
$VOL(2, 5)$	45.40872	0.0000000
$VOL(3, 5)$	48.62891	0.0000000
$VOL(4, 2)$	23.48955	0.0000000
$VOL(5, 2)$	20.89957	0.0000000
$VOL(6, 1)$	100.0000	0.0000000
$VOL(7, 1)$	39.10010	0.0000000
$VOL(8, 1)$	27.06643	0.0000000
$VOL(8, 5)$	0.4547474E-12	0.0000000
$VOL(9, 1)$	20.31608	0.0000000
$VOL(10, 5)$	51.13144	0.0000000
$VOL(11, 4)$	33.19754	0.0000000
$VOL(12, 4)$	40.93452	0.0000000
$VOL(13, 5)$	54.01499	0.0000000
$VOL(14, 4)$	58.82350	0.0000000
$VOL(14, 5)$	0.5684342E-13	0.0000000
$VOL(15, 2)$	26.21095	0.0000000
$VOL(15, 3)$	43.78905	0.0000000
$VOL(16, 2)$	24.23657	0.2176940
$VOL(16, 4)$	17.04444	0.0000000
$VOL(17, 2)$	3.803054	0.0000000
$VOL(17, 3)$	72.03255	-0.4182476E-05
$VOL(18, 2)$	0.8881784E-15	0.1489908
$VOL(18, 3)$	57.55117	0.0000000
$VOL(19, 3)$	64.21183	0.0000000
$VOL(20, 3)$	62.41540	0.0000000

8.11 Problems

1. The Slick Oil Company is preparing to make next month's pipeline shipment decisions. The Los Angeles terminal will require 200,000 barrels of oil. This oil can be supplied from either Houston or Casper, Wyoming. Houston can supply oil to L.A. at a transportation cost of \$.25 per barrel. Casper can supply L.A. at a transportation cost of \$.28 per barrel. The St. Louis terminal will require 120,000 barrels. St. Louis can be supplied from Houston at a cost of \$.18 per barrel and from Casper at a cost of \$.22 per barrel. The terminal at Freshair, Indiana requires 230,000 barrels. Oil can be shipped to Freshair from Casper at a cost of \$.21 per barrel, from Houston at a cost of \$.19 per barrel, and from Titusville, Pa. at a cost of \$.17 per barrel. Casper will have a total of 250,000 barrels available to be shipped. Houston will have 350,000 barrels available to be shipped. Because of limited pipeline capacity, no more than 180,000 barrels can be shipped from Casper to L.A. next month and no more than 150,000 barrels from Houston to L.A. The Newark, N.J. terminal will require 190,000 barrels next month. It can be supplied only from Titusville at a cost of \$.14 per barrel. The Atlanta terminal will require 150,000 barrels next month. Atlanta can be supplied from Titusville at a cost of \$.16 per barrel or from Houston at a cost of \$.20 per barrel. Titusville will have a total of 300,000 barrels available to be shipped.

Formulate the problem of finding the minimum transportation cost distribution plan as a linear program.

2. Louis Szathjoseph, proprietor of the Boulangerie Restaurant, knows he will need 40, 70, and 60 tablecloths on Thursday, Friday, and Saturday, respectively, for scheduled banquets. He can rent tablecloths for three days for \$2 each. A tablecloth must be laundered before it can be reused. He can have them cleaned overnight for \$1.50 each. He can have them laundered by regular one-day service (e.g., one used on Thursday could be reused on Saturday) for \$.80 each. There are currently 20 clean tablecloths on hand with none dirty or at the laundry. Rented tablecloths need not be cleaned before returning.
- What are the decision variables?
 - Formulate the LP appropriate for minimizing the total cost of renting and laundering the tablecloths. For each day, you will probably have a constraint requiring the number of clean tablecloths available to at least equal that day's demand. For each of the first two days, you will probably want a constraint requiring the number of tablecloths sent to the laundry not to exceed those that have been made dirty. Is it a network LP?
3. The Millersburg Supply Company uses a large fleet of vehicles it leases from manufacturers. The following pattern of vehicle requirements is forecast for the next 8 months:

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Vehicles Required	430	410	440	390	425	450	465	470

Vehicles can be leased from various manufacturers at various costs and for various lengths of time. The best plans available are: three-month lease, \$1,700; four-month lease, \$2,000; five-month lease, \$2,600. A lease can be started in any month. On January 1, there are 200 cars on lease, all of which go off lease at the end of February.

- Formulate an approach for minimizing Millersburg's leasing costs over the 8 months.
- Show that this problem is a network problem.

4. Several years ago, a university in the Westwood section of Los Angeles introduced a bidding system for assigning professors to teach courses in its business school. The table below describes a small, slightly simplified three-professor/two-course version. For the upcoming year, each professor submits a bid for each course and places limits on how many courses he or she wants to teach in each of the school's two semesters. Each professor, however, is expected to teach four courses total per year (at most three per semester).

	Prof. X	Prof. Y	Prof. Z	
Fall Courses	≤ 1	≤ 3	≤ 1	Sections Needed in the Year
Spring Courses	≤ 3	≤ 2	≤ 3	
				Min Max
Course A bids	6	3	8	3 7
Course B bids	4	7	2	2 8

From the table, note that: professor *Z* strongly prefers to teach course *A*; whereas, professor *X* has a slight preference for *A*. Professor *Y* does not want to teach more than two course sections in the Spring. Over both semesters, at least three sections of Course *A* must be taught. Can you formulate this problem as a network problem?

5. *Aircraft Fuel Ferrying Problem.* Fuel cost is one of the major components of variable operating cost for an airline. Some cities collect a tax on aircraft fuel sold at their airports. Thus, the cost per liter of fuel may vary noticeably from one airport to another. A standard problem with any airliner is the determination of how much fuel to take on at each stop. Fuel consumption is minimized if just sufficient fuel is taken on at each stop to fly the plane to the next stop. This policy, however, disregards the fact that fuel prices may differ from one airport to the next. Buying all the fuel at the cheapest stop may not be the cheapest policy either. This might require carrying large fuel loads that would in turn cause large amounts of fuel to be burned in ferrying the fuel. The refueling considerations at a given stop on a route are summarized by the following three numbers: (a) the minimum amount of fuel that must be on board at takeoff to make it to the next stop, (b) the cost per liter of fuel purchased at this stop, and (c) the amount of additional fuel above the minimum that is burned per liter of fuel delivered to the next stop. These figures are given below for an airplane that starts at Dallas, goes to Atlanta, then Chicago, Des Moines, St. Louis, and back to Dallas.

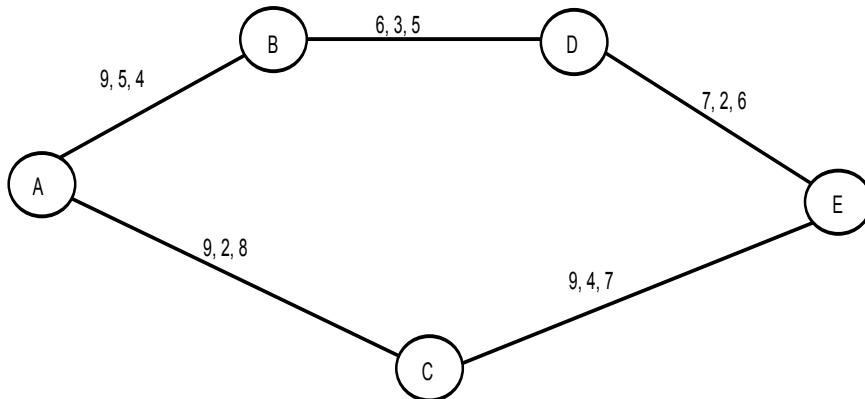
	Dallas	Atlanta	Chicago	Des Moines	St. Louis
a)	3100	2700	1330	1350	2500
b)	.29	.34	.35	.31	.33
c)	.04	.03	.02	.01	.04

For example to fly from Dallas to Atlanta, the plane must take off with at least 3100 liters of fuel. Any fuel purchased in Dallas costs \$0.29 per liter. For each liter of fuel delivered to Atlanta (i.e., still in the tank), an additional .04 liters had to be put in at Dallas. Alternatively, each additional 1.04 liters loaded at Dallas, results in an additional liter delivered to Atlanta. The plane has a maximum fuel carrying capacity of 6000 liters, which we will assume is independent of airport. Also, assume the minimum amount of reserve fuel that must be on board for safety reasons is fixed independent of airport, so we can act as if no reserve is required.

Formulate and solve a model for deciding how much fuel to buy at each airport. Is this problem any form of a network LP?

6. Show that any LP can be converted to an equivalent LP in which every column (variable) has at most three nonzero constraint coefficients. What does this suggest about the fundamental complexity of a network LP vs. a general LP?
7. Figure 8.12 is the activity-on-arc diagram showing the precedence relations among the five activities involved in repairing a refinery. The three numbers above each arc represent (from left to right, respectively) the normal time for performing the activity in days, the time to perform the activity if crashed to the maximum extent, and the additional cost in \$1000s for each day the activity is shortened. An activity can be partially crashed. It is desired the project be completed in 15 days. Write an LP formulation for determining how much each activity should be crashed.

Figure 8.12 PERT Diagram with Crashing Allowed



8. Given n currencies, the one-period currency exchange problem is characterized by a beginning inventory vector, an exchange rate matrix, and an ending inventory requirement vector defined as follows:

n_i = amount of cash on hand in currency i , at the beginning of the period measured in units of currency i , for $i = 1, 2, \dots, n$;

r_{ij} = units of currency j obtainable per unit of currency i for $i = 1, 2, \dots, n, j = 1, 2, \dots, n$. Note that $r_{ii} = 1$ and, in general, we can expect $r_{ij} < 1/r_{ji}$, for $i \neq j$.

e_i = minimum ending inventory requirement for currency i , for $i = 1, 2, \dots, n$. That is, at the end of the period, we must have at least e_i units of currency i on hand.

The decision variables are:

X_{ij} = amount of currency i converted into currency j , for $i = 1, 2, \dots, n; j = 1, 2, \dots, n$.

Formulate a model for determining an efficient set of values for the X_{ij} . The formulation should have the following features:

- a) If there is a “money pump” kind of arbitrage opportunity, the model will find it.
 - b) It should not be biased against any particular currency (i.e., the solution should be independent of which currency is called 1).
 - c) If a currency is worthless, you should buy no more of it than sufficient to meet the minimum requirement. A currency i is worthless if $r_{ij} = 0$, for all $j \neq i$.
9. The following linear program happens to be a network LP. Draw the corresponding network. Label the nodes and links.

```

MIN = 4 * T + 2 * U + 3 * V + 5 * W + 6 * X + 7 * Y + 9 * Z;
[A] T + Y + Z >= 4;
[B] U - W - X - Z = 0;
[C] - T + W = 1;
[D] V + X - Y = 2;
[E] U + V <= 7;
END

```

10. Consider a set of three flights provided by an airline to serve four cities, A , B , C , and H . The airline uses a two-fare pricing structure. The decision of how many seats or capacity to allocate to each price class is sometimes called yield or revenue management. We would like to decide upon how many seats to allocate to each fare class on each flight. Node H is a hub for changing planes. The three flights are: from A to H , H to B , and H to C . The respective flight capacities are 120, 100, and 110. Customer demand has the following characteristics:

Itinerary	Class 1 Demand	At a Price of	Class 2 Demand	At a Price of
AH	33	190	56	90
AB (via H)	24	244	43	193
AC (via H)	12	261	67	199
HB	44	140	69	80
HC	16	186	17	103

How many seats should be allocated to each class on each of the three flights? An obvious solution, if it is feasible, is to set aside enough class 1 seats on every flight, so all class 1 travelers can be accommodated. Thus, the leg AH would get $33 + 24 + 12 = 69$ class 1 seats, leg HB would get $24 + 44 = 68$ class 1 seats and leg HC would get $12 + 16 = 28$ class 1 seats. The total revenue of this solution is \$38,854. Is this the most profitable solution?

11. A common distribution system structure in many parts of the world is the three-level system composed of plants, distribution centers (DC), and outlets. A cost minimization model for a system composed of two plants (*A* & *B*), three DC's (*X*, *Y*, and *Z*), and four outlets (1, 2, 3, and 4) is shown below:

```

MIN = AX + 2 * AY + 3 * BX + BY + 2 * BZ + 5 * X1 +
      7 * X2 + 9 * Y1 + 6 * Y2 + 7 * Y3 + 8 * Z2 + 7
      * Z3 + 4 * Z4;
AX + AY = 9;
BX + BY + BZ = 8;
- AX - BX + X1 + X2 = 0;
- AY - BY + Y1 + Y2 + Y3 = 0;
- BZ + Z2 + Z3 + Z4 = 0;
- X1 - Y1 = - 3;
- X2 - Y2 - Z2 = - 5;
- Y3 - Z3 = - 4;
- Z4 = - 5;
END

```

Part of the solution is shown below:

Variable	Value	Reduced Cost
AX	3.000000	0.0000000
AY	6.000000	0.0000000
BX	0.0000000	3.000000
BY	3.000000	0.0000000
BZ	5.000000	0.0000000
X1	3.000000	0.0000000
X2	0.0000000	0.0000000
Y1	0.0000000	5.000000
Y2	5.000000	0.0000000
Y3	4.000000	0.0000000
Z2	0.0000000	3.000000
Z3	0.0000000	1.000000
Z4	5.000000	0.0000000

- a) Is there an alternate optimal solution to this distribution problem?
- b) A trucking firm that offers services from city *Y* to city 1 would like to get more of your business. At what price per unit might you be willing to give them more business according to the above solution?
- c) The demand at city 2 has been decreased to 3 units. Show how the model is changed.
- d) The capacity of plant *B* has been increased to 13 units. Show how the model is changed.
- e) Distribution center *Y* is actually in a large city where there is an untapped demand of 3 units that could be served directly from the DC at *Y*. Show how to include this additional demand at *Y*.

12. Labor on the first shift of a day (8 a.m. to 4 p.m.) costs \$15 per person × hour. Labor on the second (4 p.m. to midnight) and third (midnight to 8 a.m.) shifts cost \$20 per person × hour and \$25 per person × hour, respectively. A certain task requires 18 days if done with just first shift labor and costs \$8640. Second and third shift labor has the same efficiency as first shift labor. The only way of accelerating or crashing the task is to add additional shifts for one or more additional days. The total cost of the task consists solely of labor costs.

Complete the following crash cost table for this task.

Task time in whole days	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4
Total cost	8640														

13. You are on a camping trip and wish to prepare a recipe for a certain food delight that calls for 4 cups of water. The only containers in your possession are two ungraduated steel vessels, one of 3-cup capacity, the other of 5-cup capacity. Show how you can solve this problem by drawing a certain two-dimensional network, where each node represents a specific combination of contents in your two containers.

14. Following is part of the schedule for an airline:

Flight	City		Time		Difference in Profit	
	Depart	Arrive	Depart	Arrive		
1	221	ORD	DEN	0800	0934	+\$3000
2	223	ORD	DEN	0900	1039	-\$4000
3	274	LAX	DEN	0800	1116	-\$3000
4	105	ORD	LAX	1100	1314	+\$10000
5	228	DEN	ORD	1100	1423	-\$2000
6	230	DEN	ORD	1200	1521	-\$3000
7	259	ORD	LAX	1400	1609	+\$4000
8	293	DEN	LAX	1400	1510	+\$1000
9	412	LAX	ORD	1400	1959	+\$7000
10	766	LAX	DEN	1600	1912	+\$2000
11	238	DEN	ORD	1800	2121	-\$4000

The airline currently flies the above schedule using standard Boeing 737 aircraft. Boeing is trying to convince the airline to use a new aircraft, the 737-XX, known affectionately as the Dos Equis. The 737-XX consumes more fuel per kilometer. However, it is sufficiently larger such that, if it carries enough passengers, it is more efficient per passenger kilometer. The “Difference in Profit” column above shows the relative profitability of using the 737-XX instead of the standard 737 on each flight. The airline is considering using at most one 737-XX.

Based on the available information, analyze the wisdom of using the 737-XX in place of one of the standard 737's.

15. The following linear program happens to be a network LP:

```

MIN = 9 * S + 4 * T + 2 * U + 3 * V + 5 * W + 6 * X + 7 * Y;
[A] - T + W = 1;
[B] S + T + Y >= 4;
[C] U - W - X - S = 0;
[D] U + V <= 7;
[E] V + X - Y = 2;
END

```

- a) Draw the corresponding network.
- b) Label the nodes and links.

16. A small, but growing, long-distance phone company, SBG Inc., is trying to decide in which markets it should try to expand. It has used the following model to decide how to maximize the calls it carries per hour:

```

MAX = P1SEADNV + P2SEADNV + P1SEACHI + P2SEACHI
      + P1SEAATL + P2SEAATL + P1SEAMIA + P2SEAMIA
      + P1DNVCHI + P2DNVCHI + P1DNVATL + P2DNVATL
      + P1DNVMIA + P2DNVMIA + P1CHIATL + P2CHIATL
      + P1CHIMIA + P2CHIMIA + P1ATLMIA;
[LSEADNV] P1SEADNV + P2SEACHI + P2SEAATL + P2SEAMIA
      + P1DNVCHI + P2DNVATL + P2DNVMIA + P2CHIATL +
      P2CHIMIA <= 95;
[LSEACHI] P2SEADNV + P1SEACHI + P1SEAATL + P1SEAMIA
      + P1DNVCHI + P2DNVATL + P2DNVMIA + P2CHIATL +
      P2CHIMIA <= 80;
[LDNVATL] P2SEADNV + P2SEACHI + P2SEAATL + P2SEAMIA
      + P2DNVCHI + P1DNVATL + P1DNVMIA + P2CHIATL +
      P2CHIMIA <= 200;
[LCHIATL] P2SEADNV + P2SEACHI + P1SEAATL + P1SEAMIA
      + P2DNVCHI + P2DNVATL + P2DNVMIA + P1CHIATL +
      P1CHIMIA <= 110;
[LATLMIA] P1SEAMIA + P2SEAMIA + P1DNVMIA + P2DNVMIA
      + P1CHIMIA + P2CHIMIA + P1ATLMIA <= 105;
[DSEADNV] P1SEADNV + P2SEADNV <= 10;
[DSEACHI] P1SEACHI + P2SEACHI <= 20;
[DSEAATL] P1SEAATL + P2SEAATL <= 38;
[DSEAMIA] P1SEAMIA + P2SEAMIA <= 33;
[DDNVCHI] P1DNVCHI + P2DNVCHI <= 42;
[DDNVATL] P1DNVATL + P2DNVATL <= 48;
[DDNVMIA] P1DNVMIA + P2DNVMIA <= 23;
[DCHIATL] P1CHIATL + P2CHIATL <= 90;
[DCHIMIA] P1CHIMIA + P2CHIMIA <= 36;
[DATLMIA] P1ATLMIA <= 26;

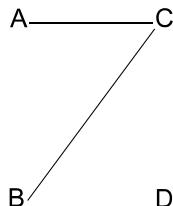
```

Now, it would like to refine the model, so it takes into account not only revenue per call, but also modest variable costs associated with each link for carrying a call. The variable cost per typical call according to link used is shown in the table below:

Variable Cost/Call				
	DNV	CHI	ATL	MIA
SEA	.11	.16	X	X
DNV		X	.15	X
CHI			.06	X
ATL				.07

An X means there is no direct link between the two cities. SBG would like to find the combination of calls to accept to maximize profit contribution. Suppose the typical revenue per call between *ATL* and *SEA* is \$1.20. Show how to modify the model just to represent the revenue and cost information for the demand between *SEA* and *ATL*.

17. Below is a four-activity project network presented in activity-on-node form, along with information on crashing opportunities for each activity:



Activity	Normal Time (days)	Crash Cost Per Day	Minimum Possible Time (days)
A	8	3	4
B	7	4	5
C	6	6	3
D	9	2	5

Complete the following tabulation of crashing cost vs. project length:

Step	Project Length	Incremental Crashing Cost/Day	Total Crashing Cost	Activities to Crash
0	16	0	0	—
1				
2				
3				
4				

Multi-period Planning Problems

9.1 Introduction

One of the most important uses of optimization is in multi-period planning. Most of the problems we have considered thus far have been essentially one-period problems. The formulations acted as if decisions this period were decoupled from decisions in future periods. Typically, however, if we produce more of a certain product this period than required by a constraint, that extra production will not be worthless, but can probably be used next period.

These interactions between periods can be represented very easily within optimization models. In fact, most large linear programs encountered in practice are multi-period models. A common synonym for “multi-period” is “dynamic” (e.g., a multi-period LP may be referred to as a dynamic model).

In some applications, the need to represent the multi-period aspects is quite obvious. One setting in which multi-period LP has been used for a number of years is in the manufacture of cheese. Production decisions must be made monthly or even weekly. The production time for many cheeses, however, may be months. For example, Parmesan cheese may need to be stored in inventory for up to ten months. What Americans call Swiss cheese may take from two to four months. The various grades of cheddar obtained depend upon the number of weeks held in storage. Sharp cheddar may be aged up to a year in storage. Clearly, in such applications, the multi-period aspect of the model is the important feature.

Models for planning over time represent the real world by partitioning time into a number of periods. The portion of the model corresponding to a single period might be some combination of product mix, blending, and other models. These single-period or static models are linked by:

1. A link or inventory variable for each commodity and period. The linking variable represents the amount of commodity transferred from one period to the next.
2. A “material balance” or “sources = uses” constraint for each commodity and period. The simplest form of this constraint is “beginning inventory + production = ending inventory + goods sold”.

Multi-period models are usually used in a rolling or sliding format. In this format, the model is solved at the beginning of each period. The recommendations of the solution for the first period are implemented. As one period elapses and better data and forecasts become available, the model is slid

forward one period. The period that had been number 2 becomes number 1, etc., and the whole process is repeated. When using a model in this sliding fashion, a practical problem is that, as the new information becomes available, this period's "optimal" solution may be drastically different from the previous period's "optimal" solution. The people who have to implement the solution may find this disconcerting. The scheduling system is said to suffer from nervousness. An approach that has been used successfully in scheduling ships, scheduling plant closings/openings, and scheduling production of breakfast cereal, see Brown, Dell, and Wood (1997), is to specify a "reference" solution (e.g., the previous period's solution). One then defines a secondary objective of minimizing the deviation of the current solution from the reference solution. If one puts zero weight on the secondary objective, then one gets the theoretically optimal solution. If one puts an extremely high weight on the secondary objective, then one simply gets the reference solution returned. If a modest weight is placed on the secondary objective, then one gets a solution that is a good compromise between low cost as measured by standard accounting, but is also close to the reference solution.

There is nothing sacred about having all periods of the same length. For example, when a petroleum company plans production for the coming year, it is sensible to have the periods correspond to the seasons of the year. One possible partition is to have the winter period extend from December 1 to March 15, the spring period extend from March 16 to May 15, the summer period extend from May 16 to September 15, and the autumn period extend from September 16 to November 30.

Some companies, such as forest product or mineral resource based companies, plan as much as 50 years into the future. In such a case, one might have the first two periods be one year each, the next period be two years, the next two periods three years each, the next two periods five years each, and the final three periods ten years each.

Inter-period interactions are usually accounted for in models by the introduction of inventory decision variables. These variables "link" adjacent periods. As an example, suppose we have a single explicit decision to make each period. Namely, how much to produce of a single product. Call this decision variable for period j , P_j . Further, suppose we have contracts to sell known amounts of this product, d_j , in period j . Define the decision variable I_j as the amount of inventory left over at the end of period j . By this convention, the beginning inventory in period j is I_{j-1} . The LP formulation will then contain one "sources of product = uses of product" constraint for each period. For period 2, the sources of product are beginning inventory, I_1 , and production in the period, P_2 . The uses of product are demand, d_2 , and end of period inventory, I_2 . For example, if $d_2 = 60$ and $d_3 = 40$, then the constraint for period 2 is:

$$I_1 + P_2 = 60 + I_2 \quad \text{or} \quad I_1 + P_2 - I_2 = 60.$$

The constraint for period 3 is:

$$I_2 + P_3 - I_3 = 40.$$

Notice how I_2 "links" (i.e., appears in both the constraints for periods 2 and 3).

In some problems, the net outflow need not exactly equal the net inflow into the next period. For example, if the product is cash, then one of the linking variables may be short-term borrowing or lending. For each dollar carried over from period 2 by lending, we will enter period 3 with \$1.05 if the interest rate is 5% per period.

On the other hand, if the "product" is workforce and there is a predictable attrition rate of 10% per period, then the above two constraints would be modified to:

$$.90I_1 + P_2 - I_2 = 60$$

$$.90I_2 + P_3 - I_3 = 40.$$

In this case, P_i is the number hired in period i .

The following example provides a simplified illustration of a single-product, multi-period planning situation.

9.2 A Dynamic Production Problem

A company produces one product for which the demand for the next four quarters is predicted to be:

Spring	Summer	Autumn	Winter
20	30	50	60

Assuming all the demand is to be met, there are two extreme policies that might be followed:

1. “Track” demand with production and carry no inventory.
2. Produce at a constant rate of 40 units per quarter and allow inventory to absorb the fluctuations in demand.

There are costs associated with carrying inventory and costs associated with varying the production level, so one would expect the least-cost policy is probably a combination of (1) and (2) (i.e., carry some inventory, but also vary the production level somewhat).

For costing purposes, the company estimates changing the production level from one period to the next costs \$600 per unit. These costs are often called “hiring and firing” costs. It is estimated that charging \$700 for each unit of inventory at the end of the period can accurately approximate inventory costs. The initial inventory is zero and the initial production level is 40 units per quarter. We require these same levels be achieved or returned to at the end of the winter quarter.

We can now calculate the production change costs associated with the no-inventory policy as:

$$\$600 \times (20 + 10 + 20 + 10 + 20) = \$48,000.$$

On the other hand, the inventory costs associated with the constant production policy is:

$$\$700 \times (20 + 30 + 20 + 0) = \$49,000.$$

The least cost policy is probably a mix of these two pure policies. We can find the least-cost policy by formulating a linear program.

9.2.1 Formulation

The following definitions of variables will be useful:

P_i = number of units produced in period i , for $i = 1, 2, 3$, and 4 ;

I_i = units in inventory at the end of period i ;

U_i = increase in production level between period $i - 1$ and i ;

D_i = decrease in production level between $i - 1$ and i .

The P_i variables are the obvious decision variables. It is useful to define the I_i , U_i , and D_i variables, so we can conveniently compute the costs each period.

To minimize the cost per year, we want to minimize the sum of inventory costs:

$$\$700 I_1 + \$700 I_2 + \$700 I_3 + \$700 I_4$$

plus production change costs:

$$\begin{aligned} \$600 U_1 + \$600 U_2 + \$600 U_3 + \$600 U_4 + \$600 U_5 \\ + \$600 D_1 + \$600 D_2 + \$600 D_3 + \$600 D_4 + \$600 D_5. \end{aligned}$$

We have added a U_5 and a D_5 in order to charge for the production level change back to 40, if needed at the end of the 4th period.

9.2.2 Constraints

Every multi-period problem will have a “material balance” or “sources = uses” constraint for each product per period. The usual form of these constraints in words is:

$$\text{beginning inventory} + \text{production} - \text{ending inventory} = \text{demand}.$$

Algebraically, these constraints for the problem at hand are:

$$\begin{aligned} P_1 - I_1 &= 20 \\ I_1 + P_2 - I_2 &= 30 \\ I_2 + P_3 - I_3 &= 50 \\ I_3 + P_4 - I_4 &= 60 \end{aligned}$$

Notice I_4 and I_0 do not appear in the first and last constraints, because initial and ending inventories are required to be zero.

If the formulation is solved as is, there is nothing to force U_1 , D_1 , etc., to be greater than zero. Therefore, the solution will be the pure production policy. Namely, $P_1 = 20$, $P_2 = 30$, $P_3 = 50$, $P_4 = 60$. This policy implies a production increase at the end of every period, except the last. This suggests a way of forcing U_1 , U_2 , U_3 , and U_4 to take the proper values is to append the constraints:

$$\begin{aligned} U_1 &\geq P_1 - 40 \\ U_2 &\geq P_2 - P_1 \\ U_3 &\geq P_3 - P_2 \\ U_4 &\geq P_4 - P_3. \end{aligned}$$

Production decreases are still not properly measured. An analogous set of four constraints should take care of this problem, specifically:

$$\begin{aligned} D_1 &\geq 40 - P_1 \\ D_2 &\geq P_1 - P_2 \\ D_3 &\geq P_2 - P_3 \\ D_4 &\geq P_3 - P_4. \end{aligned}$$

To incorporate the requirement that the production level be returned to 40 at the end of the winter quarter, we add the variables U_5 and D_5 to measure changes at the end of the last quarter. U_5 and D_5 are forced to take on the right values with the constraints:

$$\begin{aligned} U_5 &\geq 40 - P_4 \\ D_5 &\geq P_4 - 40. \end{aligned}$$

Before moving on, we will note the production-change constraints can be reduced to 5 constraints from the 10 implied by the above form. The key observation is two constraints such as:

$$\begin{aligned} U_2 &\geq P_2 - P_1 \\ D_2 &\geq P_1 - P_2 \end{aligned}$$

can be replaced by the single constraint:

$$U_2 - D_2 = P_2 - P_1.$$

The argument is more economic than algebraic. The purpose with either formulation is to force $U_2 = P_2 - P_1$ if $P_2 - P_1 \geq 0$ and $D_2 = P_1 - P_2$ if $P_1 - P_2 \geq 0$. From economics, you can argue that, at the optimal solution, you will find at most one of U_2 and D_2 are greater than 0 under either formulation. If both U_2 and D_2 are greater than 0 under the second formulation, then both can be reduced by an equal amount. Thus, reducing costs without violating any constraints.

The complete formulation is:

```
MODEL:
!Minimize inventory + workforce change costs;
MIN = 700 * I1 + 700 * I2 + 700 * I3 + 700 * I4
      + 600 * U1 + 600 * U2 + 600 * U3 + 600 * U4
      + 600 * D1 + 600 * D2 + 600 * D3 + 600 * D4
      + 600 * U5 + 600 * D5;
!Initial conditions on inventory & production;
[CNDBI] I0 = 0;
[CNDBP] P0 = 40;
!Beginning inventory + production = demand + ending inventory;
[INV1] I0 + P1 = 20 + I1;
[INV2] I1 + P2 = 30 + I2;
[INV3] I2 + P3 = 50 + I3;
[INV4] I3 + P4 = 60 + I4;
!Change up - change down = prod. this period - prod. prev. period;
[CHG1] U1 - D1 = P1 - P0;
[CHG2] U2 - D2 = P2 - P1;
[CHG3] U3 - D3 = P3 - P2;
[CHG4] U4 - D4 = P4 - P3;
[CHG5] U5 - D5 = P5 - P4;
!Ending conditions;
[CNDEI] I4 = 0;
[CNDEP] P5 = 40;
END
```

The solution is:

Optimal solution found at step:	7
Objective value:	43000.00
Variable	Value
I1	5.000000
I2	0.0000000
I3	5.000000
I4	0.0000000
U1	0.0000000
U2	0.0000000
U3	30.00000
	Reduced Cost
I1	0.0000000
I2	200.0000
I3	0.0000000
I4	0.0000000
U1	1200.000
U2	250.0000
U3	0.0000000

U4	0.0000000	250.0000
D1	15.00000	0.0000000
D2	0.0000000	950.0000
D3	0.0000000	1200.000
D4	0.0000000	950.0000
U5	0.0000000	1200.000
D5	15.00000	0.0000000
I0	0.0000000	0.0000000
P0	40.00000	0.0000000
P1	25.00000	0.0000000
P2	25.00000	0.0000000
P3	55.00000	0.0000000
P4	55.00000	0.0000000
P5	40.00000	0.0000000
Row	Slack or Surplus	Dual Price
1	43000.00	-1.000000
CNDBI	0.0000000	-950.0000
CNDBP	0.0000000	-600.0000
INV1	0.0000000	950.0000
INV2	0.0000000	250.0000
INV3	0.0000000	-250.0000
INV4	0.0000000	-950.0000
CHG1	0.0000000	600.0000
CHG2	0.0000000	-350.0000
CHG3	0.0000000	-600.0000
CHG4	0.0000000	-350.0000
CHG5	0.0000000	600.0000
CNDEI	0.0000000	-1650.000
CNDEP	0.0000000	600.0000

We see the solution is a mixed policy:

$$P_1 = P_2 = 25; \quad P_3 = P_4 = 55.$$

The mixed policy found by LP is \$5,000 cheaper than the best pure policy.

9.2.3 Representing Absolute Values

You may be tempted to represent the production-change costs in the above model by the expression:

$$600 * (@ABS(P1 - P0) + @ABS(P2 - P1) + ... + @ABS(P5 - P4));$$

This is mathematically correct, but computationally unwise, because it converts a linear program into a nonlinear program. Nonlinear programs are always more time consuming to solve. We have exploited the following result to obtain a linear program from an apparently nonlinear program. Subject to a certain condition, any appearance in a model of a term of the form:

$$@ABS(\text{expression})$$

can be replaced by the term $U + D$, if we add the constraint:

$$U - D = \text{expression}.$$

The “certain condition” is that the model must be such that a small value of $@ABS(expression)$ is preferred to a large value for $@ABS(expression)$. The result is, if $expression$ is positive, then U will be equal to $expression$, whereas, if $expression$ is negative, then D will equal the negative of $expression$.

9.3 Multi-period Financial Models

In most multi-period planning problems, the management of liquid or cash-like assets is an important consideration. If you are willing to consider cash holdings as an inventory just like an inventory of any other commodity, then it is a small step to incorporate financial management decisions into a multi-period model. The key feature is, for every period, there is a constraint that effectively says, “sources of cash – uses of cash = 0”. The following simple, but realistic, example illustrates the major features of such models.

9.3.1 Example: Cash Flow Matching

Suppose, as a result of a careful planning exercise, you have concluded that you will need the following amounts of cash for the current plus next 14 years to meet certain commitments:

Year:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Cash (in \$1,000s)	10	11	12	14	15	17	19	20	22	24	26	29	31	33	36

A common example where such a projection is made is in a personal injury lawsuit. Both parties may reach an agreement that the injured party should receive a stream of payments such as above or its equivalent. Other examples where the above approach has been used is in designing bond portfolios to satisfy cash needs for a pension fund, or for so-called balance sheet defeasance where one kind of debt is replaced by another having the same cash flow stream.

For administrative simplicity in the personal injury example, both parties prefer an immediate single lump sum payment that is “equivalent” to the above stream of 15 payments. The party receiving the lump sum will argue that the lump sum payment should equal the present value of the stream using a low interest rate such as that obtained in a very low risk investment (i.e., a government guaranteed savings account). For example, if an interest rate of 4% is used, the present value of the stream of payments is \$230,437. The party that must pay the lump sum, however, would like to argue for a much higher interest rate. To be successful, such an argument must include evidence that such higher interest rate investments are available and are no riskier than savings accounts. The investments usually offered are government securities. Generally, a broad spectrum of such investments is available on a given day. For simplicity, assume there are just two such investments available with the following features:

Security	Current Cost	Yearly Return	Years to Maturity	Principal Repayment at Maturity
1	\$980	\$60	5	\$1000
2	\$965	\$65	12	\$1000

The paying party will offer a lump sum now with a recommendation of how much should be invested in securities 1 and 2 and in savings accounts, such that the yearly cash requirements are met with the minimum lump sum payment.

The following decision variables are useful in solving this problem:

- B_1 = amount invested now in security 1, measured in “face value amount”,
- B_2 = amount invested now in security 2, measured in “face value amount”,
- S_i = amount invested into a savings account in year i , and
- L = initial lump sum.

The objective function will be to minimize the initial lump sum. There will be a constraint for each year that forces the cash flows to net to zero. If we assume idle cash is invested at 4 percent in a savings account and all amounts are measured in \$1000’s, then the formulation is:

```

MIN = L;
L - 0.98 * B1 - 0.965 * B2 - S0 = 10;
0.06 * B1 + 0.065 * B2 + 1.04 * S0 - S1 = 11;
0.06 * B1 + 0.065 * B2 + 1.04 * S1 - S2 = 12;
0.06 * B1 + 0.065 * B2 + 1.04 * S2 - S3 = 14;
0.06 * B1 + 0.065 * B2 + 1.04 * S3 - S4 = 15;
1.06 * B1 + 0.065 * B2 + 1.04 * S4 - S5 = 17;
0.065 * B2 + 1.04 * S5 - S6 = 19;
0.065 * B2 + 1.04 * S6 - S7 = 20;
0.065 * B2 + 1.04 * S7 - S8 = 22;
0.065 * B2 + 1.04 * S8 - S9 = 24;
0.065 * B2 + 1.04 * S9 - S10 = 26;
0.065 * B2 + 1.04 * S10 - S11 = 29;
1.065 * B2 + 1.04 * S11 - S12 = 31;
1.04 * S12 - S13 = 33;
1.04 * S13 - S14 = 36;
    
```

The PICTURE of the constraint coefficients gives a better appreciation of the structure of the problem. An A represents numbers bigger than 1.0, but less than 10.0. Numbers 10 or larger, but less than 100.0, are represented by a B . Numbers less than 1.0, but at least 0.1, are represented by a T . Numbers less than 0.1, but at least 0.01, are represented by a U :

		S S S S S	
	B B S S S S S S S S S S S S S		
	L 1 2 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4		
1:	1		MIN
2:	1-T-T-1		= A
3:	U U A-1		= B
4:	U U A-1		= B
5:	U U A-1		= B
6:	U U A-1		= B
7:	A U A-1		= B
8:	U A-1		= B
9:	U A-1		= B
10:	U A-1		= B
11:	U A-1		= B
12:	U A-1		= B
13:	U A-1		= B
14:	A A-1		= B
15:		A-1	= B
16:		A-1	= B

Notice in row 7, B_1 has a coefficient of 1.06. This represents the principal repayment of \$1000 plus the interest payment of \$60 measured in \$1000's. Variable S_{14} (investment of funds in a savings account after the final payment is made) appears in the problem even though at first you might think it useless to allow such an option. S_{14} is effectively a surplus cash variable in the final period. Nevertheless, it is not unusual for the solution that minimizes the lump sum payment to have cash left over at the end of the period. This is because a bond may be the most economical way of delivering funds to intermediate periods. This may cause the big principal repayment at the end of a bond's life to "overpay" the most distant periods. The solution is:

Optimal solution found at step:	14	
Objective value:	195.6837	
Variable	Value	Reduced Cost
L	195.6837	0.0000000
B1	95.79577	0.0000000
B2	90.15474	0.0000000
S0	4.804497	0.0000000
S1	5.604481	0.0000000
S2	5.436464	0.0000000
S3	3.261727	0.0000000
S4	0.0000000	0.1069792
S5	90.40358	0.0000000
S6	80.87978	0.0000000
S7	69.97503	0.0000000
S8	56.63409	0.0000000
S9	40.75951	0.0000000
S10	22.24994	0.0000000
S11	0.0000000	0.1412458
S12	65.01479	0.0000000
S13	34.61538	0.0000000
S14	0.0000000	0.3796368

Of the \$195,683.70 lump sum payment, \$10,000 goes to immediate requirements, \$4,804.50 goes into a savings account, and $0.98 \times 95,795.77 + 0.965 \times 90,154.74 = \$180,879.20$ goes into longer-term securities. Considering a wide range of investments rather than just savings accounts has reduced the amount of the lump sum payment by about \$34,750, or 15%.

In actual solutions, one may find a major fraction of the lump sum is invested in a single security. For example, appending the following constraint limits the amount invested in security 1 to half the initial lump sum:

$$0.98 B_1 - 0.5 L \leq 0.$$

An additional complication may arise due to integrality requirements on the B_1 and B_2 investments. For example, bonds can be bought only in \$1000 increments. Generally, with a modest amount of judgment, the fractional values can be rounded to neighboring integer values with no great increase in lump sum payment. For example, if B_1 and B_2 are set to 96 and 90 in the previous example, the total cost increases to \$195,726.50 from \$195,683.70. When this is done, S_{14} becomes nonzero. Specifically, the last period is overpaid by about \$40.

A sets version that places an integrality requirement on the bond purchase variables is:

```

MODEL:
! Name= PBOND, Bond portfolio/ cash matching problem: Given cash
needs in each future period, what collection of bonds should we buy
to cover these needs?;
SETS:
BOND/1..2/ : MATAT, ! Matures at period;
              PRICE, ! Purchase price;
              CAMNT, ! Coupon payout each period;
              BUY; ! Amount to buy of each bond;
PERIOD/1..15/:
    NEED, ! Cash needed each period;
SINVEST; ! Short term investment each period;
ENDSETS
DATA:
STRTE = .04;           ! Short term interest rate;
MATAT = 6, 13;         ! Years to maturity;
PRICE = .980, .965; ! Purchase price in thousands;
CAMNT = .060, .065; ! Coupon amount in thousands;
NEED = 10, 11, 12, 14, 15, 17, 19, 20, 22, 24,
      26, 29, 31, 33, 36; ! Cash needed in
      thousands;
ENDDATA
!-----
MIN = LUMP;
! First period is slightly special;
LUMP =
NEED(1) + SINVEST( 1 ) + @SUM( BOND: PRICE * BUY);
! For subsequent periods;
@FOR( PERIOD( I)| I #GT# 1:
  @SUM( BOND( J)| MATAT( J) #GE# I:
    CAMNT( J) * BUY( J)) +
  @SUM( BOND( J)| MATAT( J) #EQ# I: BUY( J)) +
  ( 1 + STRTE) * SINVEST( I - 1) =
  NEED( I) + SINVEST( I);
);
! Can only buy integer bonds;
@FOR( BOND( J): @GIN( BUY( J)););
END

Optimal solution found at step:          28
Objective value:                      195.7265
Branch count:                         3
Variable          Value       Reduced Cost
  STRTE          0.4000000E-01      0.0000000
  LUMP            195.7265        0.0000000
  MATAT( 1)       6.000000        0.0000000
  MATAT( 2)       13.000000       0.0000000
  PRICE( 1)       0.9800000        0.0000000
  PRICE( 2)       0.9650000        0.0000000
  CAMNT( 1)       0.6000000E-01      0.0000000
  CAMNT( 2)       0.6500000E-01      0.0000000
  BUY( 1)          96.000000       0.7622063

```

BUY(2)	90.00000	0.7290568
NEED(1)	10.00000	0.0000000
NEED(2)	11.00000	0.0000000
NEED(3)	12.00000	0.0000000
NEED(4)	14.00000	0.0000000
NEED(5)	15.00000	0.0000000
NEED(6)	17.00000	0.0000000
NEED(7)	19.00000	0.0000000
NEED(8)	20.00000	0.0000000
NEED(9)	22.00000	0.0000000
NEED(10)	24.00000	0.0000000
NEED(11)	26.00000	0.0000000
NEED(12)	29.00000	0.0000000
NEED(13)	31.00000	0.0000000
NEED(14)	33.00000	0.0000000
NEED(15)	36.00000	0.0000000
SINVEST(1)	4.796526	0.0000000
SINVEST(2)	5.598387	0.0000000
SINVEST(3)	5.432322	0.0000000
SINVEST(4)	3.259615	0.0000000
SINVEST(5)	0.0000000	0.8548042
SINVEST(6)	90.61000	0.0000000
SINVEST(7)	81.08440	0.0000000
SINVEST(8)	70.17778	0.0000000
SINVEST(9)	56.83489	0.0000000
SINVEST(10)	40.95828	0.0000000
SINVEST(11)	22.44661	0.0000000
SINVEST(12)	0.1944784	0.0000000
SINVEST(13)	65.05226	0.0000000
SINVEST(14)	34.65435	0.0000000
SINVEST(15)	0.4052172E-01	0.0000000

9.4 Financial Planning Models with Tax Considerations

The next example treats a slightly more complicated version of the portfolio selection problem and then illustrates how to include and examine the effect of taxes. Winston-Salem Development Management (WSDM) is trying to complete its investment plans for the next three years. Currently, WSDM has two million dollars available for investment. At six-month intervals over the next three years, WSDM expects the following income stream from previous investments: \$500,000 (six months from now); \$400,000; \$380,000; \$360,000; \$340,000; and \$300,000 (at the end of third year). There are three development projects in which WSDM is considering participating. The Foster City Development would, if WSDM participated fully, have the following cash flow stream (projected) at six-month intervals over the next three years (negative numbers represent investments, positive numbers represent income): -\$3,000,000; -\$1,000,000; -\$1,800,000; \$400,000; \$1,800,000; \$1,800,000; \$5,500,000. The last figure is its estimated value at the end of three years. A second project involves taking over the operation of some old lower-middle-income housing on the condition that certain initial repairs to it be made and that it be demolished at the end of three years. The cash flow stream for this project, if participated in fully, would be: -\$2,000,000; -\$500,000; \$1,500,000; \$1,500,000; \$1,500,000; \$200,000; -\$1,000,000.

The third project, the Disney-Universe Hotel, would have the following cash flow stream (six-month intervals) if WSDM participated fully. Again, the last figure is the estimated value at the

end of the three years: $-\$2,000,000$; $-\$2,000,000$; $-\$1,800,000$; $\$1,000,000$; $\$1,000,000$; $\$1,000,000$; $\$6,000,000$. WSDM can borrow money for half-year intervals at 3.5 percent interest per half year. At most, 2 million dollars can be borrowed at one time (i.e., the total outstanding principal can never exceed 2 million). WSDM can invest surplus funds at 3 percent per half year.

Initially, we will disregard taxes. We will formulate the problem of maximizing WSDM's net worth at the end of three years as a linear program. If WSDM participates in a project at less than 100 percent, all the cash flows of that project are reduced proportionately.

9.4.1 Formulation and Solution of the WSDM Problem

Define:

- F = fractional participation in the Foster City problem;
- M = fractional participation in Lower-Middle;
- D = participation in Disney;
- B_i = amount borrowed in period i in 1000's of dollars, $i = 1, \dots, 6$;
- L_i = amount lent in period i in 1000's of dollars, $i = 1, \dots, 6$;
- Z = net worth after the six periods in 1000's of dollars.

The problem formally is then (all numbers will be measured in units of 1000):

```

MODEL;
MAX = Z; ! Max worth at end of final period;
! Uses - sources = supply of cash in each period;
3000 * F + 2000 * M + 2000 * D - B1 + L1 = 2000;
1000 * F + 500 * M + 2000 * D + 1.035 * B1 - 1.03 * L1 - B2 +
L2=500;
1800 * F - 1500 * M + 1800 * D + 1.035 * B2 - 1.03 * L2 - B3 +
L3=400;
-400 * F - 1500 * M - 1000 * D + 1.035 * B3 - 1.03 * L3 - B4 +
L4=380;
-1800 * F - 1500 * M - 1000 * D + 1.035 * B4 - 1.03 * L4 - B5 +
L5=360;
-1800 * F - 200 * M - 1000 * D + 1.035 * B5 - 1.03 * L5 - B6 +
L6=340;
Z - 5500 * F + 1000 * M - 6000 * D + 1.035 * B6 - 1.03 * L6=300;
! Borrowing limits;
B1 <= 2000;
B2 <= 2000;
B3 <= 2000;
B4 <= 2000;
B5 <= 2000;
B6 <= 2000;
! We can invest at most 100% in a project;
F <= 1;
M <= 1;
D <= 1;
END

```

Rows 4 through 17 are the cash flow constraints for each of the periods. They enforce the requirement that $uses - sources = 0$ for each period. In the initial period, for example, L_1 uses cash, whereas B_1 is a source of cash.

The solution is:

Optimal solution found at step:	11	
Objective value:	7665.179	
Variable	Value	Reduced Cost
Z	7665.179	0.0000000
F	0.7143414	0.0000000
M	0.6372096	0.0000000
D	0.0000000	452.3816
B1	1417.443	0.0000000
L1	0.0000000	0.8788487E-02
B2	2000.000	0.0000000
L2	0.0000000	0.3343139
B3	2000.000	0.0000000
L3	0.0000000	0.2509563
B4	448.4490	0.0000000
L4	0.0000000	0.5304549E-02
B5	0.0000000	0.5149997E-02
L5	2137.484	0.0000000
B6	0.0000000	0.5000029E-02
L6	3954.865	0.0000000
Row	Slack or Surplus	Dual Price
1	7665.179	1.000000
2	0.0000000	1.819220
3	0.0000000	1.757701
4	0.0000000	1.381929
5	0.0000000	1.098032
6	0.0000000	1.060900
7	0.0000000	1.030000
8	0.0000000	1.000000
9	582.5567	0.0000000
10	0.0000000	0.3274043
11	0.0000000	0.2454662
12	1551.551	0.0000000
13	2000.000	0.0000000
14	2000.000	0.0000000
15	0.2856586	0.0000000
16	0.3627904	0.0000000
17	1.000000	0.0000000

Thus, we should try to invest or buy 0.7143414 of the Foster City project, 0.6372096 of the Middle-income project, and invest nothing in the Disney Universe project. At the end of the planning horizon, our net worth should have grown to 7,665,179.

9.4.2 Interpretation of the Dual Prices

The dual price on each of the first seven constraints is the increase in net worth in the last period resulting from an extra dollar made available in the earliest period. For example, the 1.81922 indicates an extra dollar available at the start of period 1 would increase the net worth in the last period by about \$1.82.

An extra dollar in period 5 is worth \$1.0609 at the end, because all we will do with it is invest it for two periods at three percent. Thus, it will grow to $1.03 \times 1.03 = 1.0609$ at the end.

An extra dollar in period 4 will save us from borrowing a dollar that period. Thus, we will be \$1.035 richer in period 5. We have already seen the value per extra dollar in period 5, so the value of an extra dollar in period 4 is $1.035 \times 1.0609 = \$1.09803$.

The dual prices on the borrowing constraints can be reconciled with the rest of the dual prices as follows. Having an additional dollar in period 2 is worth \$1.7577. If this dollar were borrowed, then we would have to pay out \$1.035 in period 3, which would have an effective cost of 1.035×1.38193 . Thus, the net value in the last period of borrowing an extra dollar in period 2 is $1.7577 - 1.035 \times 1.38193 = 0.3274$, which agrees with the dual price on the borrowing constraint for period 2.

The effective interest rate or cost of capital, i , in any period t , can be found from the dual prices by deriving the rate at which one would be willing to borrow. Borrowing one dollar in period t would give us \$1 more in period t , but would require us to pay out $1 + i$ dollars in period $t + 1$. We must balance these two considerations. Consider period 1. An extra dollar is worth \$1.81922 at the end of period 6. Paying back $1 + i$ in period 2 would cost $(1 + i) \$1.7577$ at the end of period 6. Balancing these two:

$$1.81922 = (1 + i)1.7577.$$

Solving:

$$i = 0.035.$$

This is not surprising because we are already borrowing at that rate in period 1, but not to the limit.

Applying a similar analysis to the other periods, we get the following effective rates:

Period	i	Period	i
1	0.03500	4	0.035
2	0.27190	5	0.030
3	0.25855	6	0.030

9.5 Present Value vs. LP Analysis

A standard method for evaluating the attractiveness of a project is by computing the present value of its cash flow stream. LP analysis, as we have just illustrated, is a generalization of present value (PV) analysis. The assumptions underlying PV analysis are that money can be: a) borrowed or lent, b) without limit, c) at a single interest rate. An LP model, such as that just considered, gives exactly the same recommendation as PV analysis if the same assumptions are made. LP analysis, however, allows one to have a borrowing rate different from a lending rate; a borrowing rate or lending rate that varies from period to period; and/or an upper limit on the amount borrowed or lent at a given rate.

Like PV analysis, LP analysis avoids the ambiguity of multiple rates of return that can occur when the internal rate of return is used to evaluate a project. Consider a project that requires an initial investment of \$1 million, pays back \$2.5 million after one year, and incurs a termination cost after two years of \$1.55 million. This project has two internal rates of return. One is about 13.82% per year. The other is about 36.18% per year. Is the project attractive if our cost of capital is 11% per year? Both PV and LP analysis will (correctly) reject this project if our cost of capital is 12% per year, accept the project if our cost of capital is 24% per year, and reject the project if our cost of capital is 38% per year.

9.6 Accounting for Income Taxes

Suppose we take taxes into account. Let us consider the following simplified situation. There is a tax rate of fifty percent on profit for any period. If there is a loss in a period, eighty percent can be carried forward to the next period. (Typically, tax laws put a limit on how many years a loss can be carried forward, but eighty percent may be a good approximation.)

Taxable income for each of the prospective projects as well as all existing projects is given in the table below. Note that because of factors such as depreciation, actual net cash flow may be rather different from taxable income in a period:

Period	Project			
	Foster City	Lower-Middle Housing	Disney Universe	Existing
1	-100,000	-200,000	-150,000	0
2	-300,000	-400,000	-200,000	100,000
3	-600,000	-200,000	-300,000	80,000
4	-100,000	500,000	-200,000	76,000
5	500,000	1,000,000	500,000	72,000
6	1,000,000	100,000	800,000	68,000
7	4,000,000	-1,000,000	5,000,000	60,000

To formulate a model, in this case, we need to additionally define:

$$P_i = \text{profit in period } i, \text{ and}$$

$$C_i = \text{loss in period } i.$$

The formulation is affected in two ways. First, we must append some equations that force the P_i 's and C_i 's to be computed properly, and, secondly, terms must be added to the cash flow constraints to account for the cash expended in the payment of tax.

In words, one of the tax computation equations is:

$$\text{Profit} - \text{loss} = \text{revenue} - \text{expense} - 0.8 \times (\text{last period's loss}).$$

Algebraically, this equation for period 2 is:

$$P_2 - C_2 = 100 + 0.03L_1 - 300F - 400M - 200D - 0.035B_1 - 0.8C_1,$$

or in standard form:

$$P_2 - C_2 - 0.03L_1 + 300F + 400M + 200D + 0.035B_1 + 0.8C_1 = 100.$$

The entire formulation is:

```

MAX = Z;
!Cash flow constraints, uses-sources= 0, including the 50% tax usage;
3000*F +2000*M + 2000*D - B1 + L1 + 0.5*P1=2000;
1000*F + 500*M + 2000*D+1.035*B1-1.03*L1-B2+L2+0.5*P2= 500;
1800*F -1500*M + 1800*D+1.035*B2-1.03*L2-B3+L3+0.5*P3= 400;
-400*F -1500*M - 1000*D+1.035*B3-1.03*L3-B4+L4+0.5*P4= 380;
-1800*F -1500*M - 1000*D+1.035*B4-1.03*L4-B5+L5+0.5*P5= 360;
-1800*F - 200*M - 1000*D+1.035*B5-1.03*L5-B6+L6+0.5*P6= 340;
Z-5500*F+1000*M - 6000*D+1.035*B6-1.03*L6 +0.5*P7= 300;
! The borrowing limits;
B1 <= 2000;
B2 <= 2000;
B3 <= 2000;
B4 <= 2000;
B5 <= 2000;
B6 <= 2000;
! The investing limits;
F <= 1;
M <= 1;
D <= 1;
! Taxable Profit-Loss for each period;
100*F+ 200*M+ 150*D +P1 -C1 = 0;
300*F+ 400*M+ 200*D+0.035*B1-0.03*L1+P2+0.8*C1-C2=100;
600*F+ 200*M+ 300*D+0.035*B2-0.03*L2+P3+0.8*C2-C3= 80;
100*F- 500*M+ 200*D+0.035*B3-0.03*L3+P4+0.8*C3-C4= 76;
-500*F-1000*M- 500*D+0.035*B4-0.03*L4+P5+0.8*C4-C5= 72;
-1000*F- 100*M- 800*D+0.035*B5-0.03*L5+P6+0.8*C5-C6= 68;
-4000*F+1000*M-5000*D+0.035*B6-0.03*L6+P7+0.8*C6-C7= 60;

```

The solution is:

Objective value:		5899.975
Variable	Value	Reduced Cost
Z	5899.9750	0.0000000
F	0.4872107	0.0000000
M	1.0000000	0.0000000
D	0.0000000	945.00740
B1	1461.6320	0.0000000
L1	0.0000000	0.5111823E-02
P1	0.0000000	0.4499472
B2	2000.0000	0.0000000
L2	0.0000000	0.1960928
P2	0.0000000	0.3793084
B3	1046.9790	0.0000000
L3	0.0000000	0.3167932E-02
P3	0.0000000	0.2042549
B4	0.0000000	0.2575563E-02
L4	991.26070	0.0000000
P4	0.0000000	0.1107492
B5	0.0000000	0.2537532E-02
L5	3221.6490	0.0000000
P5	1072.6580	0.0000000

B6	0.0000000	0.2499981E-02
L6	4359.3480	0.0000000
P6	751.86020	0.0000000
P7	1139.6230	0.0000000
C1	248.72110	0.0000000
C2	696.29720	0.0000000
C3	1039.3640	0.0000000
C4	340.85670	0.0000000
C5	0.0000000	0.1091125
C6	0.0000000	0.1075000
C7	0.0000000	0.5000000
Row	Slack or Surplus	Dual Price
1	5899.9750	1.0000000
2	0.0000000	1.3218740
3	0.0000000	1.2860920
4	0.0000000	1.0678540
5	0.0000000	1.0456780
6	0.0000000	1.0302250
7	0.0000000	1.0150000
8	0.0000000	1.0000000
9	538.36780	0.0000000
10	0.0000000	0.1924019
11	953.02070	0.0000000
12	2000.0000	0.0000000
13	2000.0000	0.0000000
14	2000.0000	0.0000000
15	0.5127893	0.0000000
16	0.0000000	573.56060
17	1.0000000	0.0000000
18	0.0000000	-0.2109901
19	0.0000000	-0.2637376
20	0.0000000	-0.3296720
21	0.0000000	-0.4120900
22	0.0000000	-0.5151125
23	0.0000000	-0.5075000
24	0.0000000	-0.5000000

Notice tax considerations cause a substantial change in the solution. More funds are placed into the lower-middle income housing project, M , and fewer funds are invested in the Foster City project, F . Project M has cash flows, which help to smooth out the stream of yearly profits.

9.7 Dynamic or Multi-period Networks

Thus far we have viewed network problems mainly as either a steady state or a one period problem. For example, in a pipeline network model, the solution can be interpreted as either the flow of material that occurs continuously day after day, or as a flow that occurs for one period and then stops. In many real systems, however, we are interested in a flow that varies from period to period, i.e., we are interested in multi-period or dynamic solutions. In these multi-period flows we also want to take into account that it may take several periods for flow to travel over an arc. Example dynamic networks are: a) river systems with various dams where we are interested in the amount of water to be spilled over the dam each period so as to satisfy various criteria regarding lake and river levels, river flows, and hydroelectric generation needs. A period might be a day, an arc might be the river section from one dam to the next, and it may take several periods for water to flow from one dam to the next. b) evacuation of a threatened facility or region as part of disaster planning, where we are interested in what routes people should take so that a large number of people escape in a short amount of time. For a building evacuation, a period might be 10 seconds, an arc may be a hallway, or a stairwell from one door to the next. Each arc may have a capacity limit of how many people can enter it per period, c) fleet routing of airplanes or trucks, where each arc in the network is a movement that must be made by either a truck or an airplane. The lead time of an arc is the length of time that it takes a vehicle to traverse the arc.

To represent a dynamic network algebraically, we need to define:

Parameters:

$L(i,j)$ = lead time, in periods, for flow to travel from node i to node j in the arc from i to j ,

Variables:

x_{ijt} = flow entering arc ij at i in period t , and therefore exiting at j in period $t+L(i,j)$,

V_{jt} = inventory remaining at node j at the end of period t ,

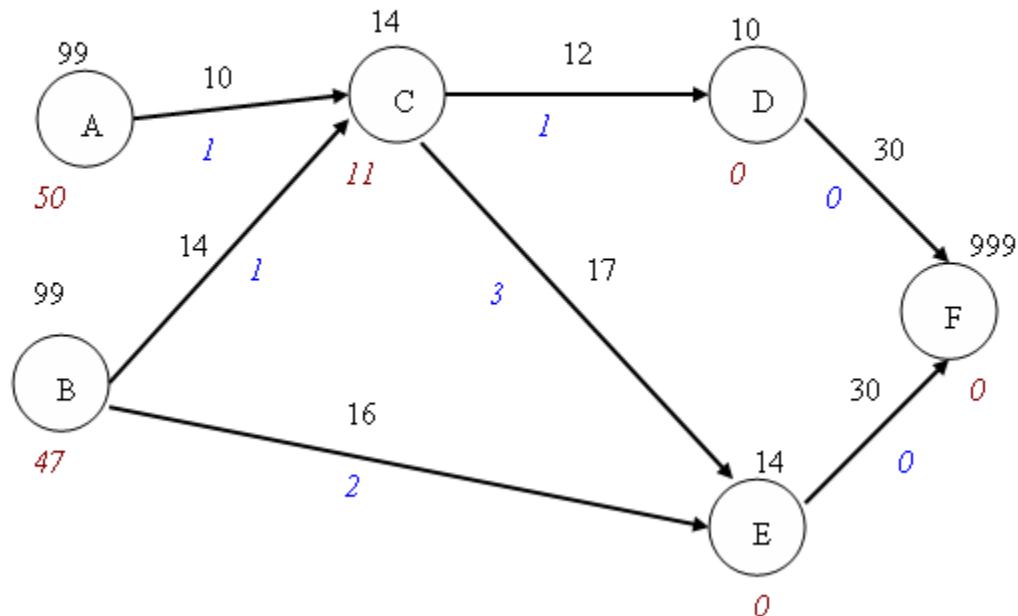
The basic node balance equation says that (inventory at node k at the end of period t) = (ending inventory at k in the preceding period) + (arriving shipments) - (shipments out of k in t), or algebraically:

$$V_{kt} = V_{kt-1} + \sum_i x_{ik(t-L(i,k))} - \sum_j x_{kjt}$$

Example:

We will illustrate the main ideas with an evacuation problem for a building. Complete details can be found in the set based LINGO model: `evacu8.lng` in the Applications Library at www.lindo.com. Figure 9.1 gives the numeric details of a building for which we want to plan evacuation routes. The nodes are places where people are or can congregate. The arcs correspond to hallways, stairwells, etc. The number of people to be evacuated from each node in the network appears in italicized font below each node. A period is a 10 second interval. The italicized number appearing below each arc is the number of periods it takes to traverse the arc. The number appearing above each arc is the upper limit on the number of people that can enter an arc per period. The number appearing above each node is the upper limit on the number of people that can be waiting at a node.

Figure 9.1 Building Evacuation Network



Node F corresponds to the outside world. For example, the fastest that one of the 50 people at node A can get to the safety of the outside world is in 2 periods by taking the path A, C, D, F. Not all the people at A can make it this fast, however, because the arc from C to D can handle only 12 people per period. Also, people from B may also try to use arc C, D. Arc (C,E) with its long lead time of three periods but relatively high capacity of 17 might correspond to a long, wide corridor, whereas arc (A,C) with its short lead time but low capacity might correspond to a short narrow stairwell.

What should be our objective? An obvious one is to minimize the number of periods that it takes to evacuate all people. An interesting challenge is to see if you can do it in at most 70 seconds for the above problem. Perhaps a more refined objective is to maximize the number of people who get out quickly. If $x_{i,j,t}$ is the number of people moving from i to j starting in period t , and realizing that node F is the outside world, we would like $x_{D,F,1} + x_{E,F,1}$ to be large, and $x_{D,F,9} + x_{E,F,9}$ to be much smaller. The objective that we will in fact use is:

$$\begin{aligned} \text{Max} = & 10*(x_{D,F,1} + x_{E,F,1}) + 9*(x_{D,F,2} + x_{E,F,2}) \\ & + 8*(x_{D,F,3} + x_{E,F,3}) + 7*(x_{D,F,4} + x_{E,F,4}) + \dots \end{aligned}$$

That is, we attach a desirable weight of 10 to getting people out in period 1, a weight of 9 to getting them out in period 2, etc. The model `evacu8.lng` is written in very general SETS form. If you want to see what the actual objective (as shown above) or constraints look like for the given data set, click on LINGO | Generate | Display model.

Suppose you allow 10 periods for our model. We do not draw the corresponding multiperiod network, but you can think of drawing it as follows. Get yourself a very wide sheet of paper and make 10 copies, side by side of the above network. Then unhook the arrow end of each arc (i, j) and reconnect it $L(i,j)$ subnetworks later. The main, nontrivial constraints in this network model are the flow balance constraints at each node each period. For example, the constraint for node E in period 5 is:

$$[\text{BAL_E_5}] - X_{B,E,3} - X_{C,E,2} + X_{E,F,5} - V_{E,4} + V_{E,5} = 0 ;$$

This is equivalent to:

$$V_{E,5} = V_{E,4} + X_{B,E,3} + X_{C,E,2} - X_{E,F,5} ;$$

In words, this says that at the end of period 5, the number of people at node E equals the number there at the end of period 4, plus people that left node B for E two periods ago, plus the number of people that left node C for E three periods ago, minus the number of people that left node E in period 5 for node F.

In general SETS form in `evacu8.lng`, this constraint is written as:

```
! For every node k and time period t;
@FOR( NXT( k,t ) | t #GT# 1:
      [BAL] V(k,t) = V(k,t-1) - @SUM( NXN(k,j) : X(k,j,t) )
            + @SUM( NXN(i,k) | t-LT(i,k) #GT# 0 : X(i,k,t-LT(i,k)) );
);
```

where the set `NXT(,)` is the set of all node k , time period t combinations, and the set `NXN(,)` is the set of all from-to arcs k,j that exist in the network.

The model is completed by adding the upper bound constraints on the number of people at each node each period, and the upper bound constraints on the number of people traveling on each arc each period. For example, the flow upper bound on the arc from B to E in period 4 is:

$$[\text{UFLO_B_E_4}] X_{B,E,4} \leq 16 ;$$

The upper bound on the number of people at node D at the end period 6 is:

$$[\text{USTOR_D_6}] V_{D,6} \leq 10 ;$$

If you solve `evacu8.lng`, you will see that you can in fact evacuate the building in 70 seconds. For simplicity and ease of direction, e.g. in terms of placement of “Exit This Way” signs, it might be desirable that the solution have all people at a given node evacuate over the same route. You may wish to check whether the solution satisfies this additional “administrative” constraint. Another example of a dynamic network, this time for a hydroelectric river system can be found in the model `dampoold.lng`. For a production example, see `mrpcap.lng`.

9.8 End Effects

Most multi-period planning models “chop” off the analysis at some finite time in the future. The manner in which this chopping off is done can be important. In general, we care about the state in which things are left at the end of a planning model (e.g., inventory levels and capital investment). If we arbitrarily terminate our planning model at year five in the future, then an optimal solution to our model may, in reality, be an optimal solution to how to go out of business in five years. Grinold (1983) provides a comprehensive discussion of various methods for mitigating end-of-horizon effects. Some of the options for handling the end effect are:

- a) *Truncation.* Simply drop from the model all periods beyond a chosen cutoff point.
- b) *Primal limits.* Place reasonable limits on things such as inventory level at the end of the final period.
- c) *Salvage values/ dual prices.* Place reasonable salvage values on things such as inventory level at the end of the final period.
- d) *Infinite final period.* Let the final period of the model represent a period of infinite length for which the same decision applies throughout the period. Net present value discounting is used in the objective function to make the final period comparable to the earlier finite periods. This is the approach used by Carino et al. (1994) in their model of the Yasuda Kasai Company, Peiser and Andrus in their model of Texas real estate development, and by Eppen, Martin, and Schrage (1988) in their model of General Motors.

9.8.1 Perishability/Shelf Life Constraints

Many products, food products in particular, are perishable. It is important to take into account the fact the product can be stored in inventory for only a modest length of time. For example, blood for blood transfusions can be stored for at most 21 days. If there is a single level of production, then this consideration is easy to represent. Define: d_t = demand in period t (given), and the variables: P_t = production in period t , and I_t = inventory at the end of period t . Then, the standard inventory balance constraint is:

$$I_{t-1} + P_t = d_t + I_t$$

If product can be carried for one period before it is discarded, then it is clear that we should add the constraint: $I_t \leq d_{t+1}$. In general, if product can be carried in inventory for at most k periods, then we add the constraint: $I_t \leq d_{t+1} + d_{t+2} \dots + d_{t+k}$.

9.8.2 Startup and Shutdown Costs

In the electric power generation industry, there is a decision problem known as the unit commitment problem. As the power demanded over the course of a day varies, the power generation company must decide which power units to start up as the demand increases and which to shutdown as demand decreases. A major concern is that there may be a significant cost to startup a generator, regardless of how long it runs. It is usually the case that the unit that is more efficient at producing power (e.g., a coal-fired unit) may, however, cost more to startup than say a gas-fired unit. Thus, if an extra burst of power is needed for only a short interval of time, it may be more cost effective to start up and run the gas-fired unit. A similar cost structure was encountered by Eppen, Martin, and Schrage(1988) in planning startup and shutdown of automotive plants. The typical way of representing startup costs, as well as shutdown costs, is with the following three sets of variables: $y_{it} = 1$ if unit i is operating in period t , else 0; $z_{it} = 1$ if unit i is started in period t , else 0; $q_{it} = 1$ if unit i stops in period t , else 0.

The crucial constraints are then:

$$z_{it} - q_{it} = y_{it} - y_{it-1}.$$

Thus, if $y_{it} = 1$, but $y_{it-1} = 0$, then z_{it} is forced to be 1. If $y_{it} = 0$, but $y_{it-1} = 1$, then q_{it} is forced to be 1. For completeness, you may also need $z_{it} + q_{it} \leq 1$, and z_{it} , q_{it} restricted to 0 or 1.

9.9 Non-optimality of Cyclic Solutions to Cyclic Problems

In some situations, such as when modeling the end of the planning horizon as above, it is reasonable to assume demand is cyclic (e.g., it repeats forever in a weekly cycle). A natural question to ask is

whether an optimal policy will have the same cycle length. We shall see that the answer may be 'no'. That is, even though demand has the same pattern, week-in and week-out, the most profitable policy need not have a weekly cycle. It may be optimal to behave differently from week to week.

In order to illustrate, let us reconsider the fleet routing and assignment problem introduced in chapter 8. We augment the original data with data on the profitability of two aircraft types for each flight:

Flight	Origin	Dest.	Depart	Arrive	Profit contribution (\$100)	
					MD90	B737
F221	ORD	DEN	800	934	115	111
F223	ORD	DEN	900	1039	109	128
F274	LAX	DEN	800	1116	129	104
F105	ORD	LAX	1100	1314	135	100
F228	DEN	ORD	1100	1423	125	102
F230	DEN	ORD	1200	1521	132	105
F259	ORD	LAX	1400	1609	112	129
F293	DEN	LAX	1400	1510	105	131
F412	LAX	ORD	1400	1959	103	135
F766	LAX	DEN	1600	1912	128	105
F238	DEN	ORD	1800	2121	128	101

For example, on flight pattern 221 an MD90 aircraft is more profitable than a B737 (\$11,500 vs. \$11,100), whereas a B737 is substantially more profitable (\$12,900 vs. \$11,200) on flight pattern 259. The above pattern of flights is to be covered every day. Suppose that we have seven MD90's available, but only one B737 available to cover these flights. As before, we assume no deadheading. First, we assume that we will use a solution with a cycle of one day. An appropriately modified model from chapter 8 is:

```

MODEL:
SETS: ! Fleet routing and assignment (FLEETRAT);
CITY :; ! The cities involved;
ACRFT: ! Aircraft types;
FCOST, ! Fixed cost per day of this type;
FSIZE; ! Max fleet size of this type;
FLIGHT:;
FXCXC( FLIGHT, CITY, CITY) :
  DEPAT, ! Flight departure time;
  ARVAT; ! arrival time at dest. ;
AXC( ACRFT, CITY):
  OVNITE; ! Number staying overnight by type, city;
AXF( ACRFT, FXCXC):
  X, ! Number aircraft used by type, flight;
  PC; ! Profit contribution by type, flight;
ENDSETS
DATA:
CITY = ORD  DEN  LAX;
ACRFT, FCOST, FSIZE =
  MD90   .01    7
  B737   .01    1;
FLIGHT = F221 F223 F274 F105 F228 F230 F259 F293 F412 F766 F238;

```

```

FXCXC, DEPAT, ARVAT =
!     Flight  Origin Dest. Depart Arrive;
      F221    ORD    DEN    800    934
      F223    ORD    DEN    900   1039
      F274    LAX    DEN    800   1116
      F105    ORD    LAX   1100   1314
      F228    DEN    ORD   1100   1423
      F230    DEN    ORD   1200   1521
      F259    ORD    LAX   1400   1609
      F293    DEN    LAX   1400   1510
      F412    LAX    ORD   1400   1959
      F766    LAX    DEN   1600   1912
      F238    DEN    ORD   1800   2121;
PC = ! Profit contribution of each vehicle*flight combo;
      115      109      129      135      125      132
      112      105      103      128      128
      111      128      104      100      102      105
      129      131      135      105      101;
ENDDATA
!-----;
! Maximize profit contribution from flights minus
overhead cost of aircraft in fleet;
MAX = @SUM( AXF( I, N, J, K): PC( I, N, J, K) * X( I, N, J, K))
      - @SUM( AXC( I, J): FCOST( I) * OVNITE( I, J));
! At any instant, departures in particular, the number of
cumulative arrivals must be >= number of cumulative departures;
! For each flight of each aircraft type;
@FOR( ACRFT( I):
@FOR( FXCXC( N, J, K):
! Aircraft on ground in morning +
number aircraft arrived thus far >=
number aircraft departed thus far;
OVNITE( I, J) +
@SUM( FXCXC( N1, J1, K1)| K1 #EQ# J #AND#
ARVAT( N1, J1, K1) #LT# DEPAT( N, J, K):
X( I, N1, J1, J)) >=
@SUM( FXCXC( N1, J1, K1)| J1 #EQ# J #AND#
DEPAT( N1, J1, K1) #LE# DEPAT( N, J, K):
X( I, N1, J, K1));
);
);
! This model does not allow deadheading, so at the end of the day,
arrivals must equal departures;
@FOR( ACRFT( I):
@FOR( CITY( J):
@SUM( AXF( I, N, J1, J): X( I, N, J1, J)) =
@SUM( AXF( I, N, J, K): X( I, N, J, K));
);
);
! Each flight must be covered;
@FOR( FXCXC( N, J, K):
@SUM( AXF( I, N, J, K): X( I, N, J, K)) = 1;
);

```

```

! Fleet size limits;
@FOR( ACRFT( I):
      @SUM( AXC( I, J): OVNITE( I, J)) <= FSIZE( I);
      );
! Fractional planes are not allowed;
@FOR( AXF: @GIN( X); );
END

```

It has the solution:

Variable	Value
X(MD90, F221, ORD, DEN)	1.000000
X(MD90, F223, ORD, DEN)	1.000000
X(MD90, F274, LAX, DEN)	1.000000
X(MD90, F105, ORD, LAX)	1.000000
X(MD90, F228, DEN, ORD)	1.000000
X(MD90, F230, DEN, ORD)	1.000000
X(MD90, F259, ORD, LAX)	1.000000
X(MD90, F412, LAX, ORD)	1.000000
X(MD90, F238, DEN, ORD)	1.000000
X(B737, F293, DEN, LAX)	1.000000
X(B737, F766, LAX, DEN)	1.000000

The daily profit contribution of this solution is $1323.94 * 100 = \$132,394$ per day. Notice that our single B737 flies from *DEN* at 2 pm to *LAX* as flight 293, and then departs *LAX* at 4 pm for *DEN* as flight 766. The above model requires that at the beginning of each day we must have the same number of MD90's and B737's at a given airport as on every other day. Just for reference, if you solve the above model with no B737's available, the profit contribution is \$132,094. So, the B737 seems to be worth only \$200 per day.

Can we do better if we allow a two-day cycle in the solution? We can try by changing the input to the model as in the model below. Effectively, we have given two days worth of demand, denoting the second day's flights by an *S*, vs. the *F* denoting the flights on the first day. Otherwise, the model is identical. The profit of this two day solution should be at least $2 * 132,394 = \$264,788$:

```

MODEL:
SETS:   ! Fleet routing and assignment (FLEETRAT);
CITY :;  ! The cities involved;
ACRFT:  ! Aircraft types;
FCOST,  ! Fixed cost per day of this type;
FSIZE;  ! Max fleet size of this type;
FLIGHT:;
FXCXC( FLIGHT, CITY, CITY) :
  DEPAT,  ! Flight departure time;
  ARVAT;  ! arrival time at dest. ;
AXC( ACRFT, CITY):
  OVNITE; ! Number staying overnight by type, city;
AXF( ACRFT, FXCXC):
  X,       ! Number aircraft used by type, flight;
  PC;     ! Profit contribution by type, flight;
ENDSETS

```

DATA:

```

CITY = ORD DEN LAX;
ACRFT, FCOST, FSIZE =
MD90 .01 7
B737 .01 1;
FLIGHT = F221 F223 F274 F105 F228 F230 F259 F293 F412 F766 F238
          S221 S223 S274 S105 S228 S230 S259 S293 S412 S766 S238;
FXCXC, DEPAT, ARVAT =
!     Flight Origin Dest. Depart Arrive;
      F221 ORD DEN 800 934
      F223 ORD DEN 900 1039
      F274 LAX DEN 800 1116
      F105 ORD LAX 1100 1314
      F228 DEN ORD 1100 1423
      F230 DEN ORD 1200 1521
      F259 ORD LAX 1400 1609
      F293 DEN LAX 1400 1510
      F412 LAX ORD 1400 1959
      F766 LAX DEN 1600 1912
      F238 DEN ORD 1800 2121
      S221 ORD DEN 3200 3334
      S223 ORD DEN 3300 3439
      S274 LAX DEN 3200 3516
      S105 ORD LAX 3500 3714
      S228 DEN ORD 3500 3823
      S230 DEN ORD 3600 3921
      S259 ORD LAX 3800 4009
      S293 DEN LAX 3800 3910
      S412 LAX ORD 3800 4359
      S766 LAX DEN 4000 4312
      S238 DEN ORD 4000 4521;
PC = ! Profit contribution of each vehicle*flight combo;
      115    109    129    135    125    132
      112    105    103    128    128
      115    109    129    135    125    132
      112    105    103    128    128
      111    128    104    100    102    105
      129    131    135    105    101
      111    128    104    100    102    105
      129    131    135    105    101;

```

ENDDATA

Now, the solution is:

Global optimal solution found at step:	103
Objective value:	2718.930
Variable	Value
X(MD90, F221, ORD, DEN)	1.000000
X(MD90, F223, ORD, DEN)	1.000000
X(MD90, F274, LAX, DEN)	1.000000
X(MD90, F105, ORD, LAX)	1.000000
X(MD90, F228, DEN, ORD)	1.000000
X(MD90, F230, DEN, ORD)	1.000000
X(MD90, F259, ORD, LAX)	1.000000
X(MD90, F293, DEN, LAX)	1.000000
X(MD90, F766, LAX, DEN)	1.000000
X(MD90, F238, DEN, ORD)	1.000000
X(MD90, S221, ORD, DEN)	1.000000
X(MD90, S274, LAX, DEN)	1.000000
X(MD90, S105, ORD, LAX)	1.000000
X(MD90, S228, DEN, ORD)	1.000000
X(MD90, S230, DEN, ORD)	1.000000
X(MD90, S259, ORD, LAX)	1.000000
X(MD90, S412, LAX, ORD)	1.000000
X(MD90, S766, LAX, DEN)	1.000000
X(MD90, S238, DEN, ORD)	1.000000
X(B737, F412, LAX, ORD)	1.000000
X(B737, S223, ORD, DEN)	1.000000
X(B737, S293, DEN, LAX)	1.000000

Notice that our profit, $2718.93 * 100 = \$271,893$ is more than twice the profit of the one day solution, $2 * 132,394 = \$264,788$. How did we arrive at this happy situation? Notice how the B737 is used. On the first day, it flies from *LAX* to *ORD* via flight 412. On the second day, it flies from *ORD* to *DEN* via flight 223 and then from *DEN* back to *LAX* via flight 293. It is only on the second day that it is back where it started, *LAX*. All three flights are very profitable for the B737 relative to the MD90. By allowing a two-day cycle, the B737 is able to cover these very profitable flights at least half of the time. Thus, even though the demand pattern has a one day cycle, it is profitable to allow the solution to have a two day cycle.

A good discussion of how to avoid the temptation to restrict solutions can be found in the book on “conceptual blockbusting” by Adams (1986). Orlin (1982) gives a more detailed analysis of the cyclic vehicle routing problem.

9.10 Problems

1. The Izza Steel Company of Tokyo has predicted delivery requirements of 3,000, 6,000, 5,000, and 2,000 tons of steel in the next four periods. Current workforce is at the 4,000 tons per period level. At the moment, there is 500 tons of steel in stock. At the end of the four periods, Izza would like its inventory position to be back at 500 tons. Regular time workforce has a variable cost of \$100 per ton. Overtime can be hired in any period at a cost of \$140 per ton. Regular time workforce size can be increased from one period to the next at a cost of \$300 per ton of change in capacity. It can be decreased at a cost of \$80 per ton. There is a charge of \$5 per ton for inventory at the end of each period. Izza would like the regular time workforce to be at the 3,000-ton level at the end of the four periods.
 - a) Formulate Izza's problem as a linear program.
 - b) What assumption does your model make about idle workforce?
2. An airline predicts the following pilot requirements for the next five quarters: 80, 90, 110, 120, 110. Current staff is 90 pilots. The question of major concern is the number of pilots to hire in each of the next five quarters. A pilot must spend the quarter in which she is hired in training. The line's training facilities limit the number of pilots in training to at most 15. Further, the training of pilots requires the services of experienced pilots at the ratio of 5 to 1 (e.g., five pilots in training require one experienced pilot). An experienced pilot so assigned cannot be used to satisfy regular requirements. The cost of hiring and training a pilot is estimated at \$20,000 exclusive of the experienced pilot time required. Experienced pilots cost \$25,000 per quarter. Company policy does not include firing pilots.
 - a) What are the variables?
 - b) Formulate a model for determining how many pilots to hire in each period.
3. The Toute de Suite Candy Company includes in its product line a number of different mixed nut products. The Chalet nut mix is required to have no more than 25 percent peanuts and no less than 40 percent almonds.

The nuts available, their prices, and their availabilities this month are as follows:

Nut	Price	Availability
Peanuts	20¢/lb.	400 lbs.
Walnuts	35¢/lb.	No limit
Almonds	50¢/lb.	200 lbs.

The Chalet mix sells for 80 cents per pound. At most, 700 pounds can be mixed per month in questions (a), (b), and (c).

- a) Formulate the appropriate model for this problem.
- b) Toute de Suite would like to incorporate into the analysis its second major mixed nut line, the Hovel line. The Hovel mix can contain no more than 60 percent peanuts and no less than 20 percent almonds. Hovel sells for 40 cents per pound. Modify your model appropriately.

- c) Toute de Suite would like to incorporate next month's requirements into the analysis. The expected situation next month is:

Nut	Price	Requirement (Availability)
Peanuts	19¢/lb.	500 lbs
Walnuts	36¢/lb.	No limit
Almonds	52¢/lb.	180 lbs.
Chalet	81¢/lb.	
Hovel	39¢/lb.	

It cost 2 cents per pound to store nuts (plain or mixed) for one month. Because of a contract commitment, at least 200 pounds of Chalet mix must be sold next month. Modify your model appropriately.

4. If two parties to a financial agreement, A and B , want the agreement to be treated as a lease for tax purposes, the payment schedule typically must satisfy certain conditions specified by the taxing agency. Suppose P_i is the payment A is scheduled to make to B in year i of a seven-year agreement. Parties A and B want to choose at the outset a set of P_i 's to satisfy a tax regulation that no payment in any given period can be less than two-thirds of the payment in any later period. Show the constraints for period i to enforce this lower bound on P_i . Use as few constraints per period as possible.
5. One of the options available to a natural gas utility is the renting of a storage facility, so it can buy gas at a cheap rate in the summer and store it until possibly needed in the winter. There is a yearly fee of \$80,000 for each year the facility is rented. There is an additional requirement that, if the utility starts renting the facility in year t , it must also rent it for at least the next three years. The gas utility has a long range planning model with a variable $x_t = 1$ if the utility rents the storage facility in year t , 0 otherwise; $y_t = 1$ if the utility starts renting in period t ; and $z_t = 1$ if the utility stops renting after period t , for $t = 1$ to 25. It is not clear whether or not this facility should be rented. Show how to represent this fee structure in an LP/IP model.
6. Below is the formulation of a cash flow matching problem, where the B variables represent investments in bonds and the S variables represent investment in savings for one period. The right-hand sides are the cash requirements for the various years.

```

MIN = L;
[P0]L -.98 * B1 -.965 * B2      - S0 = 10;
[P01] .06 * B1 + .065 * B2 + 1.04 * S0 - S1 = 11;
[P02] .06 * B1 + .065 * B2 + 1.04 * S1 - S2 = 12;
[P03] .06 * B1 + .065 * B2 + 1.04 * S2 - S3 = 14;
[P04] .06 * B1 + .065 * B2 + 1.04 * S3 - S4 = 15;
[P05] 1.06 * B1 + .065 * B2 + 1.04 * S4 - S5 = 17;
[P06]           .065 * B2 + 1.04 * S5 - S6 = 19;
[P07]           .065 * B2 + 1.04 * S6 - S7 = 20;
[P08]           .065 * B2 + 1.04 * S7 - S8 = 22;
[P09]           .065 * B2 + 1.04 * S8 - S9 = 24;
[P10]           .065 * B2 + 1.04 * S9 - S10 = 26;
[P11]           .065 * B2 + 1.04 * S10 - S11 = 29;
[P12]           1.065 * B2 + 1.04 * S11 - S12 = 31;
END

```

- a) The option to borrow at seven percent per period for a term of one period has become available in every period. Show the modification in the above model for the periods with right-hand sides of 15, 17, and 19. Denote the borrowing variables by M_0, M_1 , etc.
- b) Would this option in fact be attractive in the above model in any period?
- c) Almost all the parties involved were happy with this model until the Internal Revenue Service (IRS) suddenly became interested. The IRS has made the judgment that the initial endowment in the very first period may be tax-free. However, thereafter, the regular tax laws apply. Upon further inquiry, the IRS responded that regular income is taxed at 37 percent and capital gains at 15 percent.

We now want to find the initial lump sum such that, after taxes have been paid each period, we can still cover the right-hand side requirements. For simplicity, assume taxes are paid in the same period as the income being taxed. Show how rows P_{04} and P_{05} are altered by this unpleasant new reality (Disregard (a) and (b) above in answering.).

Blending of Input Materials

10.1 Introduction

In a blending problem, there are:

- 1) Two or more input raw material commodities;
- 2) One or more qualities associated with each input commodity;
- 3) One or more output products to be produced by blending the input commodities, so certain output quality requirements are satisfied.

A good approximation is usually that the quality of the finished product is the weighted average of the qualities of the products going into the blend.

Some examples are:

Output Commodity	Qualities	Raw Materials
Feed	Moisture, density, fraction foreign material, fraction damaged	Various types of feeds, e.g., by source.
Food	Protein, carbohydrate, fat content	Corn, oats, soybeans, meat types
Gasoline	Octane, volatility, vapor pressure	Types of crude oil refinery products
Metals	Carbon, manganese, chrome content	Metal ore, scrap metals
Grain for export	Moisture, percent foreign material, percent damaged	Grain from various suppliers
Coal for sale	Sulfur, BTU, ash, moisture content	Coal from Illinois, Wyoming, Pennsylvania
Wine	Vintage, variety, region	Pure wines of various regions
Bank balance sheet	Proportion of loans of various types, average duration of loans and investment portfolios	Types of loans and investments available

Blending models are used most frequently in three industries:

- 1) Feed and food (e.g., the blending of cattle feed, hotdogs, etc.);
- 2) Metals industry (e.g., the blending of specialty steels and nonferrous alloys, especially where recycled or scrap materials are used);
- 3) Petroleum industry (e.g., the blending of gasolines of specified octanes and volatility).

The market price of a typical raw material commodity may change significantly over the period of a month or even a week. The smart buyer will want to buy corn, for example, from the cheapest supplier. The even smarter buyer will want to exploit the fact that, as the price of corn drops relative to soybeans, the buyer may be able to save some money by switching to a blend that uses more corn.

Fields and McGee (1978) describe a feed blending LP for constructing low cost rations for cattle in a feedlot. Feedlot managers used this particular model at the rate of over 1,000 times per month. Schuster and Allen (1998) discuss the blending of grape juice at Welch's, Inc. The qualities of concern in grape juice are sweetness, acidity, and color. A blending problem must be solved at least once each season based upon how much of each type of grape is harvested by Welch's suppliers. Long term contracts require Welch's to take all of each supplier's harvest.

A recent success story in the steel industry has been the mini-mill. These small mills use mostly recyclable scrap steels to be charged into an electric furnace. The blending problem, in this case, is to decide what combination of scrap types to use to satisfy output quality requirements for specified products such as reinforcing bars, etc.

The first general LP to appear in print was a blending or diet problem formulated by George Stigler (1945). The problem was to construct a "recipe" from about 80 foods, so the mix satisfied about a dozen nutritional requirements. For example, percent protein greater than 5 percent, percent cellulose less than 40 percent, etc. When Stigler formulated this problem, the Simplex method for solving LPs did not exist. Therefore, it was not widely realized that this "diet problem" was just a special case of this wider class of problems. Stigler, realizing its generality, stated: "...there does not appear to be any direct method of finding the minimum of a linear function subject to linear conditions." The solution he obtained to his specific problem by ingenious arguments was within a few cents of the least cost solution determined later when the Simplex method was invented. Both the least cost solution and Stigler's solution were not exactly haute cuisine. Both consisted largely of cabbage, flour and dried navy beans with a touch of spinach for excitement. It is not clear that anyone would want to exist on this diet or even live with someone who was on it. These solutions illustrate the importance of explicitly including constraints that are so obvious they can be forgotten. In this case, they are palatability constraints.

10.2 The Structure of Blending Problems

Let us consider a simple feed blending problem. We must produce a batch of cattle feed having a protein content of at least 15%. Mixing corn (which is 6% protein) and soybean meal (which is 35% protein) produces this feed.

In words, the protein constraint is:

$$\frac{\text{bushels of protein in mix}}{\text{bushels in mix}} \geq 0.15$$

If C is the number of bushels of corn in the mix and S is the number of bushels of soybean meal, then we have:

$$\frac{0.06 C + 0.35 S}{C + S} \geq 0.15$$

At first glance, it looks like we have trouble. This constraint is not linear. If, however, we multiply both sides by $C + S$, we get:

$$0.06 C + 0.35 S \geq 0.15 (C + S)$$

or, in standard form, finally:

$$-0.09 C + 0.20 S \geq 0.$$

Constraints on additional characteristics (i.e., fat, carbohydrates and even such slightly nonlinear things as color, taste, and texture) can be handled in similar fashion.

The distinctive feature of a blending problem is that the crucial constraints, when written in intuitive form, are *ratios* of linear expressions. They can be converted to linear form by multiplying through by the denominator. Ratio constraints may also be found in “balance sheet” financial planning models where a financial institution may have ratio constraints on the types of loans it makes or on the average duration of its investments.

The formulation is slightly more complicated if the blending aspect is just a small portion of a larger problem in which the batch size is a decision variable. The second example in this section will consider this complication. The first example will consider the situation where the batch size is specified beforehand.

10.2.1 Example: The Pittsburgh Steel Company Blending Problem

The Pittsburgh Steel (PS) Co. has been contracted to produce a new type of very high carbon steel which has the following tight quality requirements:

	At Least	Not More Than
Carbon Content	3.00%	3.50%
Chrome Content	0.30%	0.45%
Manganese Content	1.35%	1.65%
Silicon Content	2.70%	3.00%

PS has the following materials available for mixing up a batch:

	Cost per Pound	Percent Carbon	Percent Chrome	Percent Manganese	Percent Silicon	Amount Available
Pig Iron 1	0.0300	4.0	0.0	0.9	2.25	unlimited
Pig Iron 2	0.0645	0.0	10.0	4.5	15.00	unlimited
Ferro-Silicon 1	0.0650	0.0	0.0	0.0	45.00	unlimited
Ferro-Silicon 2	0.0610	0.0	0.0	0.0	42.00	unlimited
Alloy 1	0.1000	0.0	0.0	60.0	18.00	unlimited
Alloy 2	0.1300	0.0	20.0	9.0	30.00	unlimited
Alloy 3	0.1190	0.0	8.0	33.0	25.00	unlimited
Carbide (Silicon)	0.0800	15.0	0.0	0.0	30.00	20 lb.
Steel 1	0.0210	0.4	0.0	0.9	0.00	200 lb.
Steel 2	0.0200	0.1	0.0	0.3	0.00	200 lb.
Steel 3	0.0195	0.1	0.0	0.3	0.00	200 lb.

An one-ton (2000-lb.) batch must be blended, which satisfies the quality requirements stated earlier. The problem now is what amounts of each of the eleven materials should be blended together to minimize the cost, but satisfy the quality requirements. An experienced steel man claims the least cost mix will not use any more than nine of the eleven available raw materials. What is a good blend? Most of the eleven prices and four quality control requirements are negotiable. Which prices and requirements are worth negotiating?

Note the chemical content of a blend is simply the weighted average of the chemical content of its components. Thus, for example, if we make a blend of 40% Alloy 1 and 60% Alloy 2, the manganese content is $(0.40) \times 60 + (0.60) \times 9 = 29.4$.

10.2.2 Formulation and Solution of the Pittsburgh Steel Blending Problem

The PS blending problem can be formulated as an LP with 11 variables and 13 constraints. The 11 variables correspond to the 11 raw materials from which we can choose. Four constraints are from the upper usage limits on silicon carbide and steels. Four of the constraints are from the lower quality limits. Another four constraints are from the upper quality limits. The thirteenth constraint is the requirement that the weight of all materials used must sum to 2000 pounds.

If we let P_1 be the number of pounds of Pig Iron 1 to be used and use similar notation for the remaining materials, the problem of minimizing the cost per ton can be stated as:

```

MIN =      0.03 * P1 + 0.0645 * P2 + 0.065 * F1 + 0.061 * F2 + 0.1 * A1
+ 0.13 * A2 + 0.119 * A3 + 0.08 * CB + 0.021 * S1 + 0.02 * S2 +
0.0195 * S3;
! Raw material availabilities;
CB <= 20;
S1 <= 200;
S2 <= 200;
S3 <= 200;
! Quality requirements on;
! Carbon content;
.04 * P1 + 0.15 * CB + 0.004 * S1 + 0.001 * S2 + 0.001 * S3 >= 60;
.04 * P1 + 0.15 * CB + 0.004 * S1 + 0.001 * S2 + 0.001 * S3 <= 70;
! Chrome content;
0.1 * P2 + 0.2 * A2 + 0.08 * A3 >= 6;
0.1 * P2 + 0.2 * A2 + 0.08 * A3 <= 9;
! Manganese content;
0.009 * P1 + 0.045 * P2 + 0.6 * A1 + 0.09 * A2 + 0.33 * A3 + 0.009 *
S1 + 0.003 * S2 + 0.003 * S3 >= 27;
0.009 * P1 + 0.045 * P2 + 0.6 * A1 + 0.09 * A2 + 0.33 * A3 + 0.009 *
S1 + 0.003 * S2 + 0.003 * S3 <= 33;
! Silicon content;
0.0225 * P1 + 0.15 * P2 + 0.45 * F1 + 0.42 * F2 + 0.18 * A1 + 0.3 *
A2 + 0.25 * A3 + 0.3 * CB >= 54;
0.0225 * P1 + 0.15 * P2 + 0.45 * F1 + 0.42 * F2 + 0.18 * A1 + 0.3 *
A2 + 0.25 * A3 + 0.3 * CB <= 60;
! Finish good requirements;
P1 + P2 + F1 + F2 + A1 + A2 + A3 + CB + S1 + S2 + S3 = 2000;

```

In words, the general form of this model is:

Minimize cost of raw materials

subject to

- (a) Raw material availabilities (rows 2-5)
- (b) Quality requirements (rows 6-13)
- (c) Finish good requirements (row 14)

It is generally good practice to be consistent and group constraints in this fashion.

For this particular example, when writing the quality constraints, we have exploited the knowledge that the batch size is 2000. For example, 3% of 2000 is 60, 3.5% of 2000 is 70, etc.

When solved, we get the solution:

Optimal solution found at step:		11
Objective value:		59.55629
Variable	Value	Reduced Cost
P1	1474.264	0.0000000
P2	60.00000	0.0000000
F1	0.0000000	0.1035937E-02
F2	22.06205	0.0000000
A1	14.23886	0.0000000
A2	0.0000000	0.2050311E-01
A3	0.0000000	0.1992597E-01
CB	0.0000000	0.3356920E-02
S1	200.0000	0.0000000
S2	29.43496	0.0000000
S3	200.0000	0.0000000
Row	Slack or Surplus	Dual Price
1	59.55629	1.000000
2	20.00000	0.0000000
3	0.0000000	0.1771118E-03
4	170.5650	0.0000000
5	0.0000000	0.5000000E-03
6	0.0000000	-0.1833289
7	10.00000	0.0000000
8	0.0000000	-0.2547314
9	3.000000	0.0000000
10	0.0000000	-0.1045208
11	6.000000	0.0000000
12	0.0000000	-0.9880212E-01
13	6.000000	0.0000000
14	0.0000000	-0.1950311E-01

Notice only 7 of the 11 raw materials were used.

In actual practice, this type of LP was solved on a twice-monthly basis by Pittsburgh Steel. The purchasing agent used the first solution, including the reduced cost and dual prices, as a guide in buying materials. The second solution later in the month was mainly for the metallurgist's benefit in making up a blend from the raw materials actually on hand.

Suppose we can pump oxygen into the furnace. This oxygen combines completely with carbon to produce the gas CO_2 , which escapes. The oxygen will burn off carbon at the rate of 12 pounds of carbon burned off for each 32 pounds of oxygen. Oxygen costs two cents a pound. If you reformulated the problem to include this additional option, would it change the decisions? The oxygen injection option to burn off carbon is clearly uninteresting because, in the current solution, it is the lower bound constraint rather than the upper bound on carbon that is binding. Thus, burning off carbon by itself, even if it could be done at no expense, would increase the total cost of the solution.

10.3 A Blending Problem within a Product Mix Problem

One additional aspect of blending problem formulation will be illustrated with an example in which the batch size is a decision variable. In the previous example, the batch size was specified. In the following example, the amount of product to be blended depends upon how cheaply the product can be

blended. Thus, it appears the blending decision and the batch size decision must be made simultaneously.

This example is suggestive of gasoline blending problems faced in a petroleum refinery. We wish to blend gasoline from three ingredients: butane, heavy naphtha, and catalytic reformate. Four characteristics of the resultant gasoline and its inputs are important: cost, octane number, vapor pressure, and volatility. These characteristics are summarized in the following table:

Feature	Commodity			Regular Gasoline (REG)	Premium Gasoline (PRM)
	Butane (BUT)	Catalytic Reformate (CAT)	Heavy Naphtha (NAP)		
Cost/Unit	7.3	18.2	12.5	-18.4	-22
Octane	120.0	100.0	74.0	$89 \leq \text{oct} \leq 110$	$94 \leq \text{oct} \leq 110$
Vapor Pressure	60.0	2.6	4.1	$8 \leq \text{vp} \leq 11$	$8 \leq \text{vp} \leq 11$
Volatility	105.0	3.0	12.0	$17 \leq \text{vo} \leq 25$	$17 \leq \text{vo} \leq 25$
Availability	1000.0	4000.0	5000.0	$4000 \leq \text{sell} \leq 8000$	$2000 \leq \text{sell} \leq 6000$

The cost per unit for *REG* and *PRM* are listed as negative, meaning we can sell them. That is, a negative cost is a revenue.

The octane rating is a measure of the gasoline's resistance to "knocking" or "pinging". Vapor pressure and volatility are closely related. Vapor pressure is a measure of susceptibility to stalling, particularly on an unusually warm spring day. Volatility is a measure of how easily the engine starts in cold weather.

From the table, we see in this planning period, for example, there are only 1,000 units of butane available. The profit contribution of regular gasoline is \$18.40 per unit *exclusive* of the cost of its ingredients.

A slight simplification assumed in this example is that the interaction between ingredients is linear. For example, if a "fifty/fifty" mixture of *BUT* and *CAT* is made, then its octane will be $0.5 \times 120 + 0.5 \times 100 = 110$ and its volatility will be $0.5 \times 105 + 0.5 \times 3 = 54$. In reality, this linearity is violated slightly, especially with regard to octane rating.

10.3.1 Formulation

The quality constraints require a bit of thought. The fractions of a batch of *REG* gasoline consisting of Butane, Catalytic Reformate, and Heavy Naphtha are *BUT/REG*, *CAT/REG*, and *NAP/REG*, respectively. Thus, if the god of linearity smiles upon us, the octane constraint of the blend for *REG* should be the expression:

$$(BUT/REG) \times 120 + (CAT/REG) \times 100 + (NAP/REG) \times 74 \geq 89.$$

Your expression, however, may be a frown because a ratio of variables like *BUT/REG* is definitely not linear. Multiplying through by *REG*, however, produces the linear constraint:

$$120 \text{ BUT} + 100 \text{ CAT} + 74 \text{ NAP} \geq 89 \text{ REG}$$

or in standard form:

$$120 \text{ BUT} + 100 \text{ CAT} + 74 \text{ NAP} - 89 \text{ REG} \geq 0.$$

10.3.2 Representing Two-sided Constraints

All the quality requirements are two sided. That is, they have both an upper limit and a lower limit. The upper limit constraint on octane is clearly:

$$120 \text{ BUT} + 100 \text{ CAT} + 74 \text{ NAP} - 110 \text{ REG} \leq 0.$$

We can write it in equality form by adding an explicit slack:

$$120 \text{ BUT} + 100 \text{ CAT} + 74 \text{ NAP} - 110 \text{ REG} + SOCT = 0.$$

When $SOCT = 0$, the upper limit is binding. You can verify that, when $SOCT = 110 \text{ REG} - 89 \text{ REG} = 21 \text{ REG}$, the lower limit is binding. Thus, a compact way of writing both the upper and lower limits is with the two constraints:

- 1) $120 \text{ BUT} + 100 \text{ CAT} + 74 \text{ NAP} - 110 \text{ REG} + SOCT = 0,$
- 2) $SOCT \leq 21 \text{ REG}.$

Notice, even though there may be many ingredients, the second constraint involves only two variables. This is a compact way of representing two-sided constraints.

Similar arguments can be used to develop the vapor and volatility constraints. Finally, a constraint must be appended, which states the whole equals the sum of its raw material parts, specifically:

$$REG = BUT + NAP + CAT.$$

When all constraints are converted to standard form and the expression for profit contribution is written, we obtain the formulation:

```

MODEL:
MAX = 22 * B_PRM + 18.4 * B_REG - 7.3 * XBUT_PRM - 7.3 * XBUT_REG
      - 12.5 * XNAP_PRM - 12.5 * XNAP_REG
      - 18.2 * XCAT_PRM - 18.2 * XCAT_REG;
! Subject to raw material availabilities;
[RMLIMBUT] XBUT_PRM + XBUT_REG <= 1000;
[RMLIMCAT] XCAT_PRM + XCAT_REG <= 4000;
[RMLIMNAP] XNAP_PRM + XNAP_REG <= 5000;
!For each finished good, batch size computation;
[BDEF_REG] B_REG - XNAP_REG - XCAT_REG - XBUT_REG=0;
[BDEF_PRM] B_PRM - XNAP_PRM - XCAT_PRM - XBUT_PRM=0;
! Batch size limits;
[BLO_REG] B_REG >= 4000;
[BHI_REG] B_REG <= 8000;
[BLO_PRM] B_PRM >= 2000;
[BHI_PRM] B_PRM <= 6000;
! Upper(UP) and Lower(DN) quality restrictions for each product;
[QUPREGOC] - 110 * B_REG
            + SOCT_REG + 74 * XNAP_REG + 100 * XCAT_REG + 120 * XBUT_REG = 0;
[QDNREGOC] - 21 * B_REG + SOCT_REG <= 0;
[QUPREGVA] - 11 * B_REG
            + SVAP_REG + 4.1 * XNAP_REG + 2.6 * XCAT_REG + 60 * XBUT_REG = 0;
[QDNREGVA] - 3 * B_REG + SVAP_REG <= 0;
[QUPREGVO] - 25 * B_REG
            + SVOL_REG + 12 * XNAP_REG + 3 * XCAT_REG + 105 * XBUT_REG = 0;
[QDNREGVO] - 8 * B_REG + SVOL_REG <= 0;

```

```

[QUPPRMOC] - 110 * B_PRM
    + SOCT_PRM + 74 * XNAP_PRM + 100 * XCAT_PRM + 120 * XBUT_PRM = 0;
[QDNPRMOC] - 16 * B_PRM + SOCT_PRM <= 0;
[QUPPRMVA] - 11 * B_PRM
    + SVAP_PRM + 4.1 * XNAP_PRM + 2.6 * XCAT_PRM + 60 * XBUT_PRM = 0;
[QDNPRMVA] - 3 * B_PRM + SVAP_PRM <= 0;
[QUPPRMVO] - 25 * B_PRM
    + SVOL_PRM + 12 * XNAP_PRM + 3 * XCAT_PRM + 105 * XBUT_PRM = 0;
[QDNPRMVO] - 8 * B_PRM + SVOL_PRM <= 0;
END

```

The following is the same problem, set in a general, set-based blending formulation:

```

MODEL:
    ! General Blending Model(BLEND) in LINGO;
SETS:
!Each raw material has availability & cost/unit;
    RM/ BUT, CAT, NAP/: A, C;
! Each f. g. has min & max sellable, profit
contr./unit and batch size to be determined;
    FG/ REG, PRM/: D, E, P, B;
    ! There are a set of quality measures;
    QM/ OCT, VAP, VOL/;
!Each RM & QM combo has a quality level;
    RQ( RM, QM): Q;
    !For each combo QM, FG there are upper &
lower limits on quality, slack on quality
    to be determined;
    QF( QM, FG): U, L, S;
!Each combination of RM and FG has an amount
    used, to be determined;
    RF( RM, FG): X;
ENDSETS
DATA:
A=1000, 4000, 5000;!Raw material availabilities;
C = 7.3, 18.2, 12.5;      ! R. M. costs;
Q = 120,   60, 105, !Quality parameters...;
    100,   2.6,   3,      ! R. M. by quality;
    74,   4.1,   12;
D = 4000, 2000; ! Min needed of each F.G. ;
E = 8000, 6000; !Max sellable of each F.G. ;
P = 18.4,   22; !Selling price of each F.G. ;
U = 110,   110, ! Upper limits on quality;
    11,   11,      ! Quality by F.G. ;
    25,   25;
L = 89,   94, !Lower limits on quality...;
    8,     8,      ! Quality by F.G. ;
    17,   17;
ENDDATA

```

```

!-----;
! The model;
! For each raw material, the availabilities;
@FOR( RM( I):
    [RMLIM] @SUM( FG( K): X( I, K)) < A( I);
);
@FOR( FG( K):
    !For each finished good, compute batch size;
    [BDEF] B( K) = @SUM( RM( I): X( I, K));
    ! Batch size limits;
    [BLO] B( K) > D( K);
    [BHI] B( K) < E( K);
    ! Quality restrictions for each quality;
    @FOR( QM( J):
        [QUP] @SUM( RM(I): Q(I, J) * X(I, K)) + S( J,
            K) = U( J, K) * B( K);
        [QDN] S(J, K) < (U(J, K) - L(J, K)) * B(K);
    );
);
!We want to maximize profit contribution;
[PROFIT] MAX = @SUM( FG: P * B
    - @SUM( RM( I): C( I) * @SUM( FG( K): X( I, K)));
);
END

```

As with all of our set based models, the data are well separated from the model equations. Thus, when the data change, the user need not be concerned with the model equations when updating the model.

The interesting part of the solution is:

Objective value:		48750.00
Variable	Value	Reduced Cost
B(REG)	4000.000	0.0000000
B(PRM)	4500.000	0.0000000
S(OCT, REG)	84000.00	0.0000000
S(OCT, PRM)	72000.00	0.0000000
S(VAP, REG)	1350.424	0.0000000
S(VAP, PRM)	7399.576	0.0000000
S(VOL, REG)	17500.00	0.0000000
S(VOL, PRM)	36000.00	0.0000000
X(BUT, REG)	507.4153	0.0000000
X(BUT, PRM)	492.5847	0.0000000
X(CAT, REG)	1409.958	0.0000000
X(CAT, PRM)	2590.042	0.0000000
X(NAP, REG)	2082.627	0.0000000
X(NAP, PRM)	1417.373	0.0000000
Row	Slack or Surplus	Dual Price
RMLIM(BUT)	0.0000000	27.05000
RMLIM(CAT)	0.0000000	6.65000
RMLIM(NAP)	1500.000	0.0000000
BDEF(REG)	0.0000000	-22.65000
BLO(REG)	0.0000000	-1.22500
BHI(REG)	4000.000	0.0000000
QUP(REG, OCT)	0.0000000	-0.4750000
QDN(REG, OCT)	0.0000000	0.4750000

QUP (REG, VAP)	0.0000000	0.0000000
QDN (REG, VAP)	10649.58	0.0000000
QUP (REG, VOL)	0.0000000	0.0000000
QDN (REG, VOL)	14500.00	0.0000000
BDEF (PRM)	0.0000000	-22.65000
BLO (PRM)	2500.000	0.0000000
BHI (PRM)	1500.000	0.0000000
QUP (PRM, OCT)	0.0000000	-0.4750000
QDN (PRM, OCT)	0.0000000	0.4750000
QUP (PRM, VAP)	0.0000000	0.0000000
QDN (PRM, VAP)	6100.424	0.0000000
QUP (PRM, VOL)	0.0000000	0.0000000
QDN (PRM, VOL)	0.0000000	0.0000000
PROFIT	48750.00	1.000000

The solution suggests that Premium is the more profitable product, so we sell the minimum amount of Regular required and then sell as much Premium as scarce resources, BUT and CAT, allow.

LP blending models have been a standard operating tool in refineries for years. Recently, there have been some instances where these LP models have been replaced by more sophisticated nonlinear models, which more accurately approximate the nonlinearities in the blending process. See Rigby, Lasdon, and Waren (1995), for a discussion of how Texaco does it.. For example, volatility may be represented by a logarithmic expression and octane may be represented with a polynomial like $a_1*x + a_2*x^2 + a_3*x^3 + a_4*x^4$, see Rardin(1998).

There is a variety of complications as gasoline blending models are made more detailed. For example, in high quality gasoline, the vendor may want the octane to be constant across volatility ranges in the ingredients. The reason is, if you “floor” the accelerator on a non-fuel injected automobile, a shot of raw gas is squirted into the intake. The highly volatile components of the blend will reach the combustion chamber first. If these components have low octane, you will have knocking, even though the “average” octane rating of the gasoline is high. This may be more important in a station selling gas for city driving than in a station on a cross country highway in Kansas where most driving is at a constant speed.

10.4 Proper Choice of Alternate Interpretations of Quality Requirements

Some quality features can be stated according to some measure of either goodness or, alternatively, undesirability. An example is the efficiency of an automobile. It could be stated in miles per gallon or alternatively in gallons per mile. In considering the quality of a blend of ingredients (e.g., the efficiency of a fleet of cars), it is important to identify whether it is the goodness or the badness measure which is additive over the components of the blend. The next example illustrates.

A federal regulation required the average of the miles per gallon computed over all automobiles sold by an automobile company in a specific year be at least 18 miles per gallon.

Let us consider a hypothetical case for the Ford Motor Company. Assume Ford sold only the four car types: Mark V, Ford, Granada, and Fiesta. Various parameters of these cars are listed below:

Car	Miles per Gallon	Marginal Prod. Cost	Selling Price
Fiesta	30	13,500	14,000
Granada	18	14,100	15,700
Ford	16	14,500	15,300
Mark V	14	15,700	20,000

There is some flexibility in the production facilities, so capacities may apply to pairs of car types. These limitations are:

Yearly Capacity in Units	Car Types Limited
250,000	Fiestas
2,000,000	Granadas plus Fords
1,500,000	Fords plus Mark V's

There is a sale capacity limit of 3,000,000 on the total of all cars sold. How many of each car type should Ford plan to sell?

Interpreting the mileage constraint literally results in the following formulation:

$$\begin{aligned}
 \text{MAX} = & 500*\text{FIESTA} + 1600*\text{GRANADA} + 4300*\text{MARKV} + 800*\text{FORD}; \\
 12 * \text{FIESTA} & - 4 * \text{MARKV} - 2 * \text{FORD} \geq 0; \\
 \text{FIESTA} & \leq 250; \\
 \text{GRANADA} & + \text{FORD} \leq 2000; \\
 \text{MARKV} & + \text{FORD} \leq 1500; \\
 \text{FIESTA} + \text{GRANADA} & + \text{MARKV} + \text{FORD} \leq 3000;
 \end{aligned}$$

Automobiles and dollars are measured in 1000s. Note row 2 is equivalent to:

$$\frac{30 \text{ Fiesta} + 18 \text{ Granada} + 16 \text{ Ford} + 14 \text{ Mark V}}{\text{Fiesta} + \text{Granada} + \text{Ford} + \text{Mark V}} \geq 18.$$

The solution is:

Optimal solution found at step:	1
Objective value:	6550000.
Variable	Value Reduced Cost
FIESTA	250.0000 0.0000000
GRANADA	2000.000 0.0000000
MARKV	750.0000 0.0000000
FORD	0.0000000 2950.000
Row	Slack or Surplus Dual Price
1	6550000. 1.000000
2	0.0000000 -1075.000
3	0.0000000 13400.00
4	0.0000000 1600.000
5	750.0000 0.0000000
6	0.0000000 0.0000000

Let's look more closely at this solution. Suppose each car is driven the same number of miles per year regardless of type. An interesting question is whether the ratio of the total miles driven by the above fleet divided by the number of gallons of gasoline used is at least equal to 18. Without loss, suppose each car is driven one mile. The gasoline used by a car driven one mile is $1/(\text{miles per gallon})$. Thus, if all the cars are driven the same distance, then the ratio of miles to gallons of fuel of the above fleet is $(250 + 2000 + 750)/[(250/30) + (2000/18) + (750/14)] = 17.3$ miles per gallon—which is considerably below the mpg we thought we were getting.

The first formulation is equivalent to allotting each automobile the same number of gallons and each automobile then being driven until it exhausts its allotment. Thus, the 18 mpg average is attained by having less efficient cars drive fewer miles. A more sensible way of phrasing things is in terms of gallons per mile. In this case, the mileage constraint is written:

$$\frac{\text{Fiesta}}{30} + \frac{\text{Granada}}{18} + \frac{\text{Ford}}{16} + \frac{\text{MarkV}}{14} \leq 1/18$$

$$\text{Fiesta} + \text{Granada} + \text{Ford} + \text{MarkV}$$

Converted to standard form this becomes:

$$-0.02222222 * \text{FIESTA} + 0.0069444444 * \text{FORD} + 0.015873016 * \text{MARKV} = 0;$$

When this problem is solved with this constraint, we get the solution:

Optimal solution found at step:	0
Objective value:	4830000.
Variable	Value
FIESTA	250.0000
FORD	0.0000000
MARKV	350.0000
GRANADA	2000.000

Notice the profit contribution drops noticeably under this second interpretation. The federal regulations could very easily be interpreted to be consistent with the first formulation. Automotive companies, however, wisely implemented the second way of computing fleet mileage rather than leave themselves open to later criticism of having implemented what Uncle Sam said rather than what he meant.

For reference, in 1998, the U.S. "light truck" (so-called sport utility vehicles) fleet mileage requirement was 20.7 miles per gallon, and the passenger car fleet requirement was 27.5 miles per gallon. For each tenth of a mile per gallon that a fleet falls short of the requirement, the U.S. Federal government sets a fine of \$5 per vehicle. The requirements are based on a "model year" basis. This gives a car manufacturer some flexibility if it looks like it might miss the target in a given year. For example, the manufacturer could "stop production" of a vehicle that has poor mileage, such as the big Chevy Suburban, and declare that all subsequent copies sold belong to the next model year. This may achieve the target in the current model year, but postpone the problem to the next model year.

10.5 How to Compute Blended Quality

The general conclusion is one should think carefully when one needs to compute an average performance measure for some blend or collection of things. There are at least three ways of computing averages or means when one has N observations, x_1, x_2, \dots, x_N :

Arithmetic: $(x_1 + x_2 + \dots + x_N)/N$

Geometric: $(x_1 \times x_2 \times \dots \times x_N)^{(1/N)}$

Harmonic: $1/[(1/x_1 + 1/x_2 + \dots + 1/x_N)/N]$

The arithmetic mean is appropriate for computing the mean return of the assets in a portfolio. If, however, we are interested in the average growth of a portfolio over time, we would probably want to use the geometric mean of the yearly growths rather than the arithmetic average. Consider, for example, an investment that has a growth factor of 1.5 in the first year and 0.67 in the second year (e.g., a rate of return of 50% in the first year and -33% in the second year). Most people would *not* consider the average growth to be $(1.5 + 0.67)/2 = 1.085$. The harmonic mean tends to be appropriate when computing an average rate of something, as in our car fleet example above.

A quality measure for which the harmonic mean is usually appropriate is density. For some products, such as food and feed, one may have available different ingredients each with different densities and desire a blend of these products with a specific density. Density is usually measured in weight per volume (e.g., grams per cubic centimeter). If the decision variables are measured in weight units rather than volume units, then the harmonic mean is appropriate.

10.5.1 Example

We have two ingredients, one with a density of 0.7 g/cc and the other with a density of 0.9 g/cc. If we mix together one gram of each, what is the density of the mix? Clearly, the mix has a weight of 2 grams. Its volume in cc's is $1/0.7 + 1/0.9$. Thus, its density is $2/(1/0.7 + 1/0.9) = 0.7875$ g/cc. This is less than the 0.8 we would predict if we took the arithmetic average. If we define:

X_i = grams of ingredient i in the mix,

t = target lower limit on density desired.

Then, we can write the density constraint for our little example as:

$$(X_1 + X_2)/(X_1/0.7 + X_2/0.9) \geq t,$$

or

$$(X_1 + X_2)/t \geq X_1/0.7 + X_2/0.9,$$

or

$$(1/t - 1/0.7)X_1 + (1/t - 1/0.9)X_2 \geq 0,$$

(i.e., a harmonic mean constraint).

10.5.2 Generalized Mean

One can generalize the idea just discussed by introducing a transformation $f(q)$. The interpretation is that the function $f()$ “linearizes” the quality. The basic idea is that many of the quality measures used in practice were chosen somewhat arbitrarily (e.g., why is the freezing point of water 32 degrees on the Fahrenheit scale?). So, even though a standardly used quality measure does not “blend linearly”, perhaps we can find a transformation that does. Such linearizations are common in industry. Some examples follow:

1. Rigby, Lasdon, and Waren (1995) use this idea when calculating the Reid vapor pressure of a blended gasoline at Texaco. If r_i is the Reid vapor pressure of component i of the blend, they use the transformation:

$$f(r_i) = r_i^{1.25}$$

For example, if component 1 has a vapor pressure of 80, component 2 has a vapor pressure of 100, x_i is the amount used of component i , and we want a blend with a vapor pressure of at least 90, the constraint could be written:

$$80^{1.25} \times x_1 + 100^{1.25} \times x_2 \geq 90^{1.25} \times (x_1 + x_2),$$

or

$$239.26 \times x_1 + 316.23 x_2 \geq 277.21 (x_1 + x_2).$$

2. The *flashpoint* of a chemical is the lowest temperature at which it will catch fire. Typical jet fuel has a flashpoint of around 100 degrees F. Typical heating oil has a flashpoint of at least 130 degrees F. The jet fuel used in the supersonic SR-71 jet aircraft had a flashpoint of several hundred degrees F. If p_i is the flashpoint of component i , then the transformation:

$$f(p_i) = 10^{42} (p_i + 460)^{-14.286}$$

will approximately linearize the flashpoint. Notice that $f(p_i)$ is a decreasing function of p_i , so a higher flashpoint means a lower $f(p_i)$ value.

For example, if component 1 has a flashpoint of 100, component 2 has a flashpoint of 140, x_i is the amount used of component i , and we want a blend with a flashpoint of at least 130, the constraint would be written:

$$10^{42} (100 + 460)^{-14.286} \times x_1 + 10^{42} (140 + 460)^{-14.286} \times x_2 \leq$$

$$10^{42} (130 + 460)^{-14.286} \times (x_1 + x_2),$$

or

$$548.76 \times x_1 + 204.8 x_2 \leq 260 (x_1 + x_2).$$

3. The American Petroleum Institute likes to measure the lightness of a material in “API gravity”, see Dantzig and Thapa (1997). Water has an API gravity of 10. API gravity does not blend linearly. However, the specific gravity, defined by:

$$sg = 141.5 / (API\ gravity + 131.5)$$

does blend linearly. Note, the specific gravity of a material is the weight in grams of one cubic centimeter of material.

4. The *viscosity* of a liquid is a measure, in units of centistokes, of the time it takes a standard cup volume of liquid, at 122 degrees Fahrenheit, to flow through a hole of a certain diameter. The higher the viscosity, the less quickly the liquid flows. If v_i is the viscosity of component i , then the transformation:

$$f(v_i) = \ln(\ln(v_i + .08))$$

will approximately linearize the viscosity.

For example, if component 1 has a viscosity of 5, component 2 has a viscosity of 25, x_i is the amount used of component i , and we want a blend with a viscosity of at most 20, the constraint would be written:

$$\ln(\ln(5 + .08)) \times x_1 + \ln(\ln(25 + .08)) \times x_2 \leq$$

$$\ln(\ln(20 + .08)) \times (x_1 + x_2),$$

or

$$.4857 x_1 + 1.17 x_2 \leq 1.0985(x_1 + x_2).$$

5. In the transmissivity of light through a glass fiber of length x_i , or the financial growth of an investment over a period of length x_i , or in the probability of no failures in a number of trials x_i , one may have constraints of the form: $a_1^{x_1} a_2^{x_2} \dots a_n^{x_n} \geq a_0$. This can be linearized by taking logarithms (e.g., $\ln(a_1) * x_1 + \ln(a_2) * x_2 + \dots + \ln(a_n) * x_n \geq \ln(a_0)$).

For example, if we expect stocks to have a long term growth rate of 10% per year, we expect less risky bonds to have a long term growth rate of 6% per year, we want an overall growth of 40% over five years, and x_1 and x_2 are the number of years we invest in stocks and bonds respectively over a five year period, then we want the constraint:

$$(1.10)^{x_1}(1.06)^{x_2} \geq 1.40.$$

Linearizing, this becomes:

$$\ln(1.10)x_1 + \ln(1.06)x_2 \geq \ln(1.40), \text{ or}$$

$$.09531x_1 + .05827x_2 \geq .3364,$$

$$x_1 + x_2 = 5.$$

The preceding examples apply the transformation to each quality individually. One could extend the idea even further by allowing a “matrix” transformation to several qualities together.

10.6 Interpretation of Dual Prices for Blending Constraints

The dual price for a blending constraint usually requires a slight reinterpretation in order to be useful. As an example, consider the minimum octane constraint for Premium gasoline in the model considered earlier. The constraint was effectively:

$$-94 B_PRM + 120 XBUT_PRM + 74 XNAP_PRM + 100 XCAT_PRM \geq 0.$$

The dual price of this constraint is the rate of increase in profit if the right-hand side of this constraint is increased from 0 to 1. Unfortunately, this is not a change we would ordinarily consider. More typical changes that might be entertained would be changing the octane rating from 94 to either 93 or 95. A very approximate rule for estimating the effect of changing the coefficient in row i of variable B_PRM is to compute the product of the dual price in row i and the value of variable B_PRM . For variable B_PRM and the octane constraint, this value is $-.475 * 4500 = -2137.5$. This suggests, if

the octane requirement is reduced to 93 (or increased to 95) from 94, the total profit will increase by about 2137.5 to $48750 + 2137.5 = \$50887.5$ (or decrease to $48750 - 2137.5 = \$46,612.5$). If the LP is actually re-solved with an octane requirement of 93 (or 95), the actual profit contribution changes to $\$51,000$ (or $\$46,714.29$).

This approximation can be summarized generally as follows:

If we wish to change a certain quality requirement of blend by a small amount δ , the effect on profit of this change is approximately of the magnitude $\delta \times (\text{dual price of the constraint}) \times (\text{batch size})$. For small changes, the approximation tends to underestimate profit after the change. For large changes, the approximation may err in either direction.

10.7 Fractional or Hyperbolic Programming

In blending problems, we have seen ratio constraints of the form:

$$\frac{\sum_i q_i X_i}{\sum_j X_j} \geq q_0$$

can be converted to linear form, by rewriting:

$$\sum_j q_j X_j \geq q_0 \sum_j X_j \quad \text{or} \quad \sum_j (q_j - q_0) x_j \geq 0$$

Can we handle a similar feature in the objective? That is, can a problem of the following form be converted to linear form?

$$(1) \text{ Maximize} \quad \frac{u_o + \sum_j u_j X_j}{v_o + \sum_j v_j X_j}$$

$$(2) \text{ subject to: } \sum_j a_{ij} X_j = b_i, \text{ for } i = 1, 2, \dots$$

The a_{ij} , u_0 , u_j , v_0 , and v_j are given constants. For example, we might wish to maximize the fraction of protein in a blend subject to constraints on availability of materials and other quality specifications.

We can make it linear with the following transformations:

Define:

$$r = 1/(v_0 + \sum_j v_j X_j)$$

and

$$y_j = X_j r$$

We assume $r > 0$.

Then our objective is:

$$(1') \text{ Maximize} \quad u_0 r + \sum_j u_j y_j$$

subject to:

$$r = 1/(v_0 + \sum_j v_j X_j), \text{ or}$$

$$(1.1') \quad r v_0 + \sum_j v_j y_j = 1$$

Any other constraint i of the form:

$$(2) \sum_j a_{ij} X_j = b_i$$

can be written as:

$$\sum_j a_{ij} X_j r = b_i r$$

or linear in terms of the new variables,

$$(2') \sum_j a_{ij} y_i - b_i r = 0$$

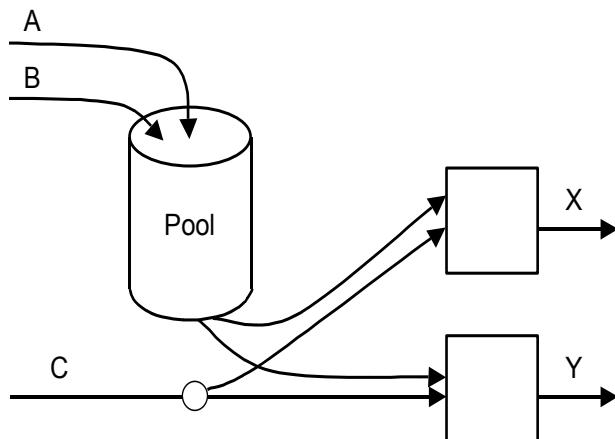
10.8 Multi-Level Blending: Pooling Problems

A complicating factor in some blending problems is that not all raw ingredients can be stored separately. Such a situation can arise in a number of ways. Two ingredients may be produced at the same location, but for economic reasons, are transported together (e.g., in one tank car or via one pipeline). Another possibility is two ingredients are delivered separately, but only a single holding facility is available at the blending site. In general, many facilities that blend ingredients have only a modest number of storage facilities. For example, a grain storage facility may have only a half dozen bins. A petroleum refinery may have only a half dozen tanks. If there are more than a half dozen different sources of raw materials, then not all raw materials can be stored separately. In the petroleum industry, this leads to what is called a pooling problem.

This pooling of raw materials within a blending problem leads to a nonlinear program. The pooling problem discussed here is taken from Haverly (1978). A , B , and C are ingredients containing 3%, 1%, and 2% sulfur as an impurity, respectively. These chemicals are to be blended to provide two output products, X and Y , which must meet sulfur content upper limits of 2.5% and 1.5%, respectively. At the given prices of \$9 per unit of X and \$15 per unit of Y , customers will buy all of X and Y produced up to a maximum of 100 units of X and 200 units of Y . The costs per unit for ingredients A , B , and C are \$6, \$16, and \$10, respectively. The problem is to operate the process in order to maximize profit.

A complicating factor in this blending process is the fact that products A and B must be stored in the same tank, or “pool”. So, until the amounts of A and B are determined, the pool sulfur content is unknown. Figure 10.1 illustrates. However, it is the pool sulfur content together with the amounts of pool material and of chemical C used in blending X and Y that determine the X and Y sulfur contents. The sulfur constraints on X and Y affect the amounts of A and B needed, and it is this “circularity” that causes a nonlinearity.

Figure 10.1 A Pooling Problem



The constraint equations defining this system involve material balances and sulfur constraints for the output products. Consider the material balance equations first.

We have the following mass balance for the pool, assuming all of the pool material is to be used up:

$$\text{Amount } A + \text{Amount } B = \text{Pool to } X + \text{Pool to } Y.$$

For the output products, the balance equations are:

$$\text{Pool to } X + \text{C to } X = \text{Amount } X$$

and:

$$\text{Pool to } Y + \text{C to } Y = \text{Amount } Y.$$

For the total amount of C, the equation is:

$$\text{C to } X + \text{C to } Y = \text{Amount } C.$$

Introducing the pool sulfur percent, Pool S, as a new variable, makes it easy to write the X and Y sulfur constraints. If we let Pool S have a value between 0 and 100 and express all other percentages on the same scale, these constraints are:

$$\text{Pool } S \times \text{Pool to } X + 2 \times C \text{ to } X \leq 2.5 \times \text{Amount } X$$

$$\text{Pool } S \times \text{Pool to } Y + 2 \times C \text{ to } Y \leq 1.5 \times \text{Amount } Y$$

The left-hand side of each inequality represents the actual sulfur content of the appropriate product and the right-hand side is the maximum amount of sulfur permitted in that product. The pool sulfur balance equation is:

$$3 \times \text{Amount } A + 1 \times \text{Amount } B = \text{Pool } S \times (\text{Amount } A + \text{Amount } B).$$

This defines Pool S as the amount of sulfur in the pool divided by the total amount of material in the pool.

As mentioned earlier, product demand sets upper bounds on production as:

$$\text{Amount } X \leq 100$$

$$\text{Amount } Y \leq 200$$

and physical considerations restrict all variables to be nonnegative quantities. Clearly, the pool sulfur can never be less than 1% or more than 3%. Thus:

$$1 \leq \text{Pool } S \leq 3.$$

Finally, the profit function must be formulated. If Cost A , Cost B , Cost C , Cost X , and Cost Y are the appropriate cost coefficients, the profit can be written as:

$$\begin{aligned} & \text{Cost } X \times \text{Amount } X + \text{Cost } Y \times \text{Amount } Y - \text{Cost } A \times \text{Amount } A \\ & - \text{Cost } B \times \text{Amount } B - \text{Cost } C \times \text{Amount } C \end{aligned}$$

A LINGO formulation follows:

```

MODEL:
COSTA = 6;
COSTB = 16;
COSTC = 10;
COSTX = 9;
COSTY = 15;
MAX = COSTX * AMOUNTX + COSTY * AMOUNTY - COSTA * AMOUNTA - COSTB *
AMOUNTB - COSTC * AMOUNTC;
! Sources = uses for the pool;
AMOUNTA + AMOUNTB = POOLTOX + POOLTOY;
! Sources for final products;
POOLTOX + CTOX = AMOUNTX;
POOLTOY + CTOY = AMOUNTY;
! Uses of C;
AMOUNTC = CTOX + CTOY;
! Blending constraints for final products;
POOLS * POOLTOX + 2 * CTOX <= 2.5 * AMOUNTX;
POOLS * POOLTOY + 2 * CTOY <= 1.5 * AMOUNTY;
! Blending constraint for the pool product;
3*AMOUNTA + AMOUNTB=POOLS*(AMOUNTA + AMOUNTB);
! Demand upper limits;
AMOUNTX <= 100;
AMOUNTY <= 200;
END

```

This problem is tricky in that it has (as we shall see) several local optima. LINGO, left to its own devices, may find the following solution:

Optimal solution found at step:	16	
Objective value:	400.0000	
Variable	Value	Reduced Cost
COSTA	6.000000	0.0000000
COSTB	16.000000	0.0000000
COSTC	10.000000	0.0000000
COSTX	9.000000	0.0000000
COSTY	15.000000	0.0000000
AMOUNTX	0.0000000	0.0000000
AMOUNTY	200.0000	0.0000000
AMOUNTA	0.0000000	2.000003
AMOUNTB	100.00000	0.0000000
AMOUNTC	100.00000	0.0000000
POOLTOX	0.0000000	4.000026
POOLTOY	100.00000	0.0000000
CTOX	0.0000000	0.0000000
CTOY	100.00000	0.0000000
POOLS	0.9999932	0.0000000

Examination of the solution shows the optimal operation produces only product Y using equal amounts of B and C. The cost per unit of output is $(16 + 10) / 2 = \$13$ and the sale price is \$15, giving a profit of \$2 per unit. Since all 200 units are produced and sold, the profit is \$400.

Nonlinear problems, such as this pooling model, have the curious feature that the solution you get may depend upon where the solver starts its solution search. You can set the starting point by inserting an "INIT" initialization section in your model such as the following:

```
INIT:
  AMOUNTX = 0;
  AMOUNTY = 0;
  AMOUNTA = 0;
  AMOUNTB = 0;
  AMOUNTC = 0;
  POOLTOX = 0;
  POOLTOY = 0;
  CTOX = 0;
  CTOY = 0;
  POOLS = 3;
ENDINIT
```

The INIT section allows you to provide the solver with an initial guess at the solution. Starting at the point provided in the INIT section, LINGO may find the solution:

Optimal solution found at step:	4
Objective value:	100.0000
Variable	Value
AMOUNTX	100.0000
AMOUNTY	0.0000000
AMOUNTA	50.00000
AMOUNTB	0.0000000
AMOUNTC	50.00000
POOLTOX	50.00000
POOLTOY	0.0000000
CTOX	50.00000
CTOY	0.0000000
POOLS	3.000000
COSTA	6.000000
COSTB	16.00000
COSTC	10.00000
COSTX	9.000000
COSTY	15.00000

In this solution, only product *X* is produced and sold. It is made using an equal blend of chemicals *A* and *C*. The net cost of production is \$8 per unit, yielding a profit of \$1 per unit of *X* sold. Since only 100 units are called for, the final profit is \$100. This solution is locally optimal. That is, small changes from this operating point reduce the profit. There are no feasible operating conditions close to this one that yield a better solution.

Our earlier solution, yielding a profit of \$400, is also a local optimum. However, there is no other feasible point with a larger profit, so we call the \$400 solution a global optimum. The reader is invited to find other local optima, for example, by increasing the use of *A* and decreasing *B* and *C*.

Generally speaking, an initial guess should not set variable values to zero. Since zero multiplied by any quantity is still zero, such values can lead to unusual behavior of the optimization algorithm. For example, if we take our previous initial guess, except set *POOLS* = 2, the solver may get stuck at this point and gives the solution:

Optimal solution found at step:	1
Objective value:	0.0000000E+00
Variable	Value
AMOUNTX	0.0000000
AMOUNTY	0.0000000
AMOUNTA	0.0000000
AMOUNTB	0.0000000
AMOUNTC	0.0000000
POOLTOX	0.0000000
POOLTOY	0.0000000
CTOX	0.0000000
CTOY	0.0000000
POOLS	2.000000
COSTA	6.000000
COSTB	16.00000
COSTC	10.00000
COSTX	9.000000

COSTY	15.00000	0.0000000
-------	----------	-----------

As this output shows, LINGO finds the starting point to be optimal. Actually, this point is not even a local optimum, but rather a stationary point (i.e., very small changes do not provide any significant improvement, within the tolerances used in the algorithm, in the objective). The point satisfies the so-called first-order necessary conditions for an optimum. If, however, the starting point is perturbed by some small amount, the solver should find an actual local optimum and perhaps the global one. In fact, setting all variables previously at zero to 0.1 does lead to the global maximum solution with profit of \$400.

This model is an example of where a global solver is helpful. If the “Global solver” option is selected in LINGO, then the global optimal solution with value 400 is found without fail. For this problem, all solutions obtained have the property that many constraints are active. In other words, they hold as equalities. Of course, the five equality constraints (rows 2 through 5 and row 8) are always active. In addition, in the globally optimal solution, the sulfur content of Y is at its upper limit, and six variables are either at lower or upper limits (*POOLS*, *CTOX*, *POOLTOX*, *AMOUNTA*, *AMOUNTY*, and *AMOUNTX*). Hence, there are twelve active constraints, but only ten variables. When there are at least as many active constraints as there are variables, this is called a vertex solution. In linear programming, any LP having an optimal solution has a vertex solution. This is not true in NLP, but vertex optima are not uncommon and seem to occur frequently in models involving blending and processing.

When there are more active constraints than variables, the vertex is called degenerate. In the global solution to this problem, there are two “extra” active constraints. One could be removed by dropping the upper and lower limits on *POOLS*. These are redundant because they are implied by constraint 8 and the nonnegativity of the variables. The lower limits on *AMOUNTX* and *AMOUNTY* could also be dropped, since they are implied by rows 3 and 4 and the lower limits on *CTOX*, *CTOY*, *POOLTOX*, and *POOLTOY*. Doing this would lead to the same vertex solution, but with exactly as many active constraints as variables. Some other constraints are redundant too. The reader is invited to find them.

10.9 Problems

1. The Exxoff Company must decide upon the blends to be used for this week’s gasoline production. Two gasoline products must be blended and their characteristics are listed below:

Gasoline	Vapor Pressure	Octane Number	Selling Price (in \$/barrel)
Lo-lead	≤ 7	≥ 80	\$ 9.80
Premium	≤ 6	≥ 100	\$12.00

The characteristics of the components from which the gasoline can be blended are shown below:

Component	Vapor Pressure	Octane Number	Available this Week (in barrels)
Cat-Cracked Gas	8	83	2700
Isopentane	20	109	1350
Straight Gas	4	74	4100

The vapor pressure and octane number of a blend is simply the weighted average of the corresponding characteristics of its components. Components not used can be sold to “independents” for \$9 per barrel.

- a) What are the decision variables?
 - b) Give the LP formulation.
 - c) How much Premium should be blended?
2. The Blendex Oil Company blends a regular and a premium product from two ingredients, Heptane and Octane. Each liter of regular is composed of exactly 50% Heptane and 50% Octane. Each liter of premium is composed of exactly 40% Heptane and 60% Octane. During this planning period, there are exactly 200,000 liters of Heptane and 310,000 liters of Octane available. The profit contributions per liter of the regular and premium product this period are \$0.03 and \$0.04 per liter respectively.
- a) Formulate the problem of determining the amounts of the regular and premium products to produce as an LP.
 - b) Determine the optimal amounts to produce without the use of a computer.
3. Hackensack Blended Whiskey Company imports three grades of whiskey: Prime, Choice, and Premium. These unblended grades can be used to make up the following two brands of whiskey with associated characteristics:

Brand	Specifications	Selling price per liter
Scottish	Not less than 60% Prime.	\$6.80
Club	Not more than 20% Premium.	
Johnny Gold	Not more than 60% Premium. Not less than 15% Prime.	\$5.70

The costs and availabilities of the three raw whiskeys are:

Whiskey	Available This Week (Number of Liters)	Cost per Liter
Prime	2,000	\$7.00
Choice	2,500	\$5.00
Premium	1,200	\$4.00

Hackensack wishes to maximize this week’s profit contribution and feels it can use linear programming to do so. How much should be made of each of the two brands? How should the three raw whiskeys be blended into each of the two brands?

4. The Sebastopol Refinery processes two different kinds of crude oil, Venezuelan and Saudi, to produce two general classes of products, Light and Heavy. Either crude oil can be processed by either of two modes of processing, Short or Regular. The processing cost and amounts of Heavy and Light produced depend upon the mode of processing used and the type of crude oil used. Costs vary, both across crude oils and across processing modes. The relevant characteristics are summarized in the table below. For example, the short process converts each unit of Venezuelan crude to 0.45 units of Light product, 0.52 units of Heavy product, and 0.03 units of waste.

	Short Process		Regular Process	
	Venezuela	Saud	Venezuela	Saud
Light product fraction	0.45	0.60	0.49	0.68
Heavy product fraction	0.52	0.36	0.50	0.32
Unusable product fraction	0.03	0.04	0.01	0.00

Saudi crude costs \$20 per unit, whereas Venezuelan crude is only \$19 per unit. The short process costs \$2.50 per unit processed, while the regular process costs \$2.10 per unit. Sebastopol can process 10,000 units of crude per week at the regular rate. When the refinery is running the Short process for the full week, it can process 13,000 units per week.

The refinery may run any combination of short and regular processes in a given week.

The respective market values of Light and Heavy products are \$27 and \$25 per unit. Formulate the problem of deciding how much of which crudes to buy and which processes to run as an LP. What are the optimal purchasing and operating decisions?

5. There has been a lot of soul searching recently at your company, the Beansoul Coal Company (BCC). Some of its better coal mines have been exhausted and it is having more difficulty selling its coal from remaining mines. One of BCC's most important customers is the electrical utility, Power to the People Company (PPC). BCC sells coal from its best mine, the Becky mine, to PPC. The Becky mine is currently running at capacity, selling all its 5000 tons/day of output to PPC. Delivered to PPC, the Becky coal costs BCC \$81/ton and PPC pays BCC \$86/ton. BCC has four other mines, but you have been unable to get PPC to buy coal from these mines. PPC says that coal from these mines does not satisfy its quality requirements. Upon pressing PPC for details, it has agreed it would consider buying a mix of coal as long as it satisfies the following quality requirements: sulfur $\leq 0.6\%$; ash $\leq 5.9\%$; BTU ≥ 13000 per ton; and moisture $\leq 7\%$. You note your Becky mine satisfies this in that its quality according to the above four measures is: 0.57%, 5.56%, 13029 BTU, and 6.2%. Your four other mines have the following characteristics:

Mine	BTU Per Ton	Sulfur Percent	Ash Percent	Moisture Percent	Cost Per Ton Delivered to PPC
Lex	14,201	0.88	6.76	5.1	73
Casper	10,630	0.11	4.36	4.6	90
Donora	13,200	0.71	6.66	7.6	74
Rocky	11,990	0.39	4.41	4.5	89

The daily capacities of your Lex, Casper, Donora, and Rocky mines are 4000, 3500, 3000, and 7000 tons respectively. PPC uses an average of about 13,000 tons per day.

BCC's director of sales was ecstatic upon hearing of your conversation with PPC. His response was "Great! Now, we will be able sell PPC all of the 13,000 tons per day it needs". Your stock with BCC's newly appointed director of productivity is similarly high. Her reaction to your discussion with PCC was: "Let's see, right now we are making a profit contribution of only \$5/ton of coal sold to PPC. I have figured out we can make a profit contribution of \$7/ton if we can sell them a mix. Wow! You are an ingenious negotiator!" What do you recommend to BCC?

6. The McClendon Company manufactures two products, bird food and dog food. The company has two departments, blending and packaging. The requirements in each department for manufacturing a ton of either product are as follows:

	Time per Unit in Tons	
	Blending	Packaging
Bird food	0.25	0.10
Dog food	0.15	0.30

Each department has 8 hours available per day.

Dog food is made from the three ingredients: meat, fishmeal, and cereal. Bird food is made from the three ingredients: seeds, ground stones, and cereal. Descriptions of these five materials are as follows.

Descriptions of Materials in Percents					
	Protein	Carbohydrates	Trace Minerals	Abrasives	Cost (in \$/ton)
Meat	12	10	1	0	600
Fishmeal	20	8	2	2	900
Cereal	3	30	0	0	200
Seeds	10	10	2	1	700
Stones	0	0	3	100	100

The composition requirements of the two products are as follows:

Composition Requirements of the Products in Percents					
	Protein	Carbohydrates	Trace Minerals	Abrasive	Seeds
Bird food	5	18	1	2	10
Dog food	11	15	1	0	0

Bird food sells for \$750 per ton while dog food sells for \$980 per ton. What should be the composition of bird food and dog food and how much of each should be manufactured each day?

7. Recent federal regulations strongly encourage the assignment of students to schools in a city, so the racial composition of any school approximates the racial composition of the entire city. Consider the case of the Greenville city schools. The city can be considered as composed of five areas with the following characteristics:

Area	Fraction Minority	Number of students
1	0.20	1,200
2	0.10	900
3	0.85	1,700
4	0.60	2,000
5	0.90	2,500

The ruling handed down for Greenville is that a school can have neither more than 75 percent nor less than 30 percent minority enrollment. There are three schools in Greenville with the following capacities:

School	Capacity
Bond	3,900
Pocahontas	3,100
Pierron	2,100

The objective is to design an assignment of students to schools, so as to stay within the capacity of each school and satisfy the composition constraints while minimizing the distance traveled by students. The distances in kilometers between areas and schools are:

School	Area				
	1	2	3	4	5
Bond	2.7	1.4	2.4	1.1	0.5
Pocahontas	0.5	0.7	2.9	0.8	1.9
Pierron	1.6	2.0	0.1	1.3	2.2

There is an additional condition that no student can be transported more than 2.6 kilometers. Find the number of students that should be assigned to each school from each area. Assume any group of students from an area has the same ethnic mix as the whole area.

8. A farmer is raising pigs for market and wishes to determine the quantity of the available types of feed that should be given to each pig to meet certain nutritional requirements at minimum cost. The units of each type of basic nutritional ingredient contained in a pound of each feed type is given in the following table along with the daily nutritional requirement and feed costs.

Nutritional Ingredient	Pound of Corn	Pound of Tankage	Pound of Alfalfa	Units Required per day
Carbohydrates	9	2	4	20
Proteins	3	8	6	18
Vitamins	1	2	6	15
Cost (cents)/lb.	7	6	5	

9. Rico-AG is a German fertilizer company, which has just received a contract to supply 10,000 tons of 3-12-12 fertilizer. The guaranteed composition of this fertilizer is (by weight) at least 3% nitrogen, 12% phosphorous, and 12% potash. This fertilizer can be mixed from any combination of the raw materials described in the table below.

Raw Material	% Nitrogen	% Phosphorous	% Potash	Current World Price/Ton
AN	50	0	0	190 Dm
SP	1	40	5	180 Dm
CP	2	4	35	196 Dm
BG	1	15	17	215 Dm

Rico-AG has in stock 500 tons of SP that was bought earlier for 220 Dm/ton. Rico-AG has a long-term agreement with Fledermausguano, S.A. This agreement allows it to buy already mixed 3-12-12 at 195 Dm/ton.

- a) Formulate a model for Rico-AG that will allow it to decide how much to buy and how to mix. State what assumptions you make with regard to goods in inventory.
 - b) Can you conclude in advance that no *CP* and *BG* will be used because they cost more than 195 Dm/ton?
10. The Albers Milling Company buys corn and wheat and then grinds and blends them into two final products, Fast-Gro and Quick-Gro. Fast-Gro is required to have at least 2.5% protein while Quick-Gro must have at least 3.2% protein. Corn contains 1.9% protein while wheat contains 3.8% protein. The firm can do the buying and blending at either the Albers (*A*) plant or the Bartelso (*B*) plant. The blended products must then be shipped to the firm's two warehouse outlets, one at Carlyle (*C*) and the other at Damiansville (*D*). Current costs per bushel at the two plants are:

	A	B
Corn	10.0	14.0
Wheat	12.0	11.0

Transportation costs per bushel between the plants and warehouses are:

Fast-Gro:		To		Quik-Gro:		To	
		C	D			C	D
From	A	1.00	2.00	From	A	3.00	3.50
	B	3.00	0.75		B	4.00	1.90

The firm must satisfy the following demands in bushels at the warehouse outlets:

Warehouse	Product	
	Fast-Gro	Quik-Gro
C	1,000	3,000
D	4,000	6,000

Formulate an LP useful in determining the purchasing, blending, and shipping decisions.

11. A high quality wine is typically identified by three attributes: (a) its vintage, (b) its variety, and (c) its region. For example, the Optima Winery of Santa Rosa, California produced a wine with a label that stated: 1984, Cabernet Sauvignon, Sonoma County. The wine in the bottle may be a blend of wines, not all of which need be of the vintage, variety, and region specified on the label. In this case, the state of California and the U.S. Department of Alcohol, Tobacco, and Firearms strictly enforce the following limits. To receive the label 1984, Cabernet Sauvignon, Sonoma County, at least 95% of the contents must be of 1984 vintage, at least 75% of the contents must be Cabernet Sauvignon, and at least 85% must be from Sonoma County. How small might be the fraction of the wine in the bottle that is of 1984 vintage *and* of the Cabernet Sauvignon variety *and* from grapes grown in Sonoma County?
12. Rogers Foods of Turlock, California (see Rosenthal and Riefel (1994)) is a producer of high quality dried foods, such as dried onions, garlic, etc. It has regularly received “Supplier of the Year” awards from its customers, retail packaged food manufacturers such as Pillsbury. A reason for Rogers’ quality reputation is it tries to supply product to its customers with quality characteristics that closely match customer specifications. This is difficult to do because Rogers does not have complete control over its input. Each food is harvested once per year from a variety of farms, one “lot” per farm. The quality of the crop from each farm is somewhat of a random variable. At harvest time, the crop is dried and each lot placed in the warehouse. Orders throughout the year are then filled from product in the warehouse.

Two of the main quality features of product are its density and its moisture content. Different customers may have different requirements for each quality attribute. If a product is too dense, then a jar that contains five ounces may appear only half full. If a product is not sufficiently dense, it may be impossible to get five ounces into a jar labelled as a five-ounce jar.

To illustrate the problem, suppose you have five lots of product with the following characteristics:

Lot	Fraction Moisture	Density	Kg. Available
1	0.03	0.80	1000
2	0.02	0.75	2500
3	0.04	0.60	3100
4	0.01	0.60	1500
5	0.02	0.65	4500

You currently have two prospective customers with the following requirements:

Customer	Fraction Moisture		Density		Max Kg. Desired	Selling Price per Kg.
	Min	Max	Min	Max		
<i>P</i>	0.035	0.045	0.70	0.75	3,000	\$5.25
<i>G</i>	0.01	0.03	0.60	0.65	15,000	\$4.25

What should you do?

13. The Lexus automobile gets 26 miles per gallon (mpg), the Corolla gets 31 mpg, and the Tercel gets 35 mpg. Let L , C , and T represent the number of automobiles of each type in some fleet. Let F represent the total number in the fleet. We require, in some sense, the mpg of the fleet to be at least 32 mpg. Fleet mpg is measured by (total miles driven by the fleet)/(total gallons of fuel consumed by fleet).
- Suppose the sense in which mpg is measured is each auto is given one gallon of fuel, then driven until the fuel is exhausted. Write appropriate constraints to enforce the 32 mpg requirement.
 - Suppose the sense in which mpg is measured is each auto is driven one mile and then stopped. Write appropriate constraints to enforce the 32 mpg requirement.
14. In the financial industry, one is often concerned with the “duration” of one’s portfolio of various financial instruments. The duration of a portfolio is simply the weighted average of the duration of the instruments in the portfolio, where the weight is simply the number of dollars invested in the instrument. Suppose the Second National Bank is considering revising its portfolio and has denoted by $X1$, $X2$, and $X3$, the number of dollars invested (in millions) in each of three different instruments. The durations of the three instruments are respectively: 2 years, 4 years, and 5 years. The following constraint appeared in their planning model:
- $$+ X1 - X2 - 2 \times X3 \geq 0$$
- In words, this constraint is:
- duration of the portfolio must be at most 10 years;
 - duration of the portfolio must be at least 3 years;
 - duration of the portfolio must be at least 2 years;
 - duration of the portfolio must be at most 3 years;
 - none of the above.
15. You are manager of a team of ditch diggers, each member of the team is characterized by a productivity measure with units of cubic feet per hour. An average productivity measure for the entire team should be based on which of the following:
- the arithmetic mean;
 - the geometric mean;
 - the harmonic mean.
16. Generic Foods has three different batches of cashews in its warehouse. The percentage moisture content for batches 1, 2, and 3 respectively are 8%, 11%, and 13%. In blending a batch of cashews for a particular customer, the following constraint appeared:
- $$+ 2 \times X1 - X2 - 3 \times X3 \geq 0$$

In words, this constraint is:

- a) percent moisture must be at most 10%;
 - b) percent moisture must be at least 3%;
 - c) percent moisture must be at least 10%;
 - d) percent moisture must be at most 2%;
 - e) none of the above.
17. The Beanbody Company buys various types of raw coal on the open market and then pulverizes the coal and mixes it to satisfy customer specifications. Last week Beanbody bought 1500 tons of type *M* coal for \$78 per ton that was intended for an order that was canceled at the last minute. Beanbody had to pay an additional \$1 per ton to have the coal shipped to its processing facility. Beanbody has no other coal in stock. Type *M* coal has a BTU content of 13,000 BTU per ton. This week type *M* coal can be bought (or sold) on the open market for \$74 per ton. Type *W* coal, which has a BTU content of 10,000 BTU/ton, can be bought this week for \$68 per ton. Type *K* coal, which has a BTU content of 12,000 BTU/ton, can be bought this week for \$71 per ton. All require an additional \$1/ton to be shipped into Beanbody's facility. In fact, Beanbody occasionally sells raw coal on the open market and then Beanbody also has to pay \$1/ton outbound shipping. Beanbody expects coal prices to continue to drop next week. Right now Beanbody has an order for 2700 tons of pulverized product having a BTU content of at least 11,000 BTU per ton. Clearly, some additional coal must be bought. The president of Beanbody sketched out the following incomplete model for deciding how much of what coal to purchase to just satisfy this order.:

MODEL:

```

! MH = tons of on-hand type M coal used;
! MP = tons of type M coal purchased;
! WP = tons of type W coal purchased;
! KP = tons of type K coal purchased;

MIN = ____ * MH + ____ * MP + ____ * WP + ____ * KP;
MH + MP + WP + KP = 2700;
MH <= 1500;
2000 * MH + ____ * MP - 1000 * WP + ____ * KP >= 0;
END

```

What numbers would you place in the _____ places?

18. A local high school is considering using an outside supplier to provide meals. The big question is: How much will it cost to provide a nutritious meal to a student? Exhibit A reproduces the recommended daily minima for an adult as recommended by the noted dietitian, George Stigler (1945). Because our high school need provide only one meal per day, albeit the main one, it should be sufficient for our meal to satisfy one-half of the minimum daily requirements.

With regard to nutritive content of foods, Exhibit B displays the nutritional content of various foods available from one of the prospective vendors recommended by a student committee at the high school. See Bosch (1993) for a comprehensive discussion of these data.

11

Formulating and Solving Integer Programs

"To be or not to be" is true.

-G. Boole

11.1 Introduction

In many applications of optimization, one would really like the decision variables to be restricted to integer values. One is likely to tolerate a solution recommending GM produce 1,524,328.37 Chevrolets. No one will mind if this recommendation is rounded up or down. If, however, a different study recommends the optimum number of aircraft carriers to build is 1.37, then a lot of people around the world will be very interested in how this number is rounded. It is clear the validity and value of many optimization models could be improved markedly if one could restrict selected decision variables to integer values.

All good commercial optimization modeling systems are augmented with a capability that allows the user to restrict certain decision variables to integer values. The manner in which the user informs the program of this requirement varies from program to program. In LINGO, for example, one way of indicating variable X is to be restricted to integer values is to put it in the model the declaration as: @GIN(X). The important point is it is straightforward to specify this restriction. We shall see later that, even though easy to specify, sometimes it may be difficult to solve problems with this restriction. The methods for formulating and solving problems with integrality requirements are called *integer programming*.

11.1.1 Types of Variables

One general classification is according to types of variables:

Pure vs. mixed. In a pure integer program, all variables are restricted to integer values. In a mixed formulation, only certain of the variables are integer; whereas, the rest are allowed to be continuous.

0/1 vs. general. In many applications, the only integer values allowed are 0/1. Therefore, some integer programming codes assume integer variables are restricted to the values 0 or 1.

The integrality enforcing capability is perhaps more powerful than the reader at first realizes. A frequent use of integer variables in a model is as a zero/one variable to represent a go/no-go decision. It is probably true that the majority of real world integer programs are of the zero/one variety.

11.2 Exploiting the IP Capability: Standard Applications

You will frequently encounter LP problems with the exception of just a few combinatorial complications. Many of these complications are fairly standard. The next several sections describe many of the standard complications along with the methods for incorporating them into an IP formulation. Most of these complications only require the 0/1 capability rather than the general integer capability. Binary variables can be used to represent a wide variety of go/no-go, or make-or-buy decisions. In the latter use, they are sometimes referred to as “Hamlet” variables as in: “To buy or not to buy, that is the question”. Binary variables are sometimes also called Boolean variables in honor of the logician George Boole. He developed the rules of the special algebra, now known as Boolean algebra, for manipulating variables that can take on only two values. In Boole’s case, the values were “True” and “False”. However, it is a minor conceptual leap to represent “True” by the value 1 and “False” by the value 0. The power of these methods developed by Boole is undoubtedly the genesis of the modern compliment: “Strong, like Boole.”

11.2.1 Binary Representation of General Integer Variables

Some algorithms apply to problems with only 0/1 integer variables. Conceptually, this is no limitation, as any general integer variable with a finite range can be represented by a set of 0/1 variables. For example, suppose X is restricted to the set $[0, 1, 2, \dots, 15]$. Introduce the four 0/1 variables: y_1, y_2, y_3 , and y_4 . Replace every occurrence of X by $y_1 + 2 \times y_2 + 4 \times y_3 + 8 \times y_4$. Note every possible integer in $[0, 1, 2, \dots, 15]$ can be represented by some setting of the values of y_1, y_2, y_3 , and y_4 . Verify that, if the maximum value X can take on is 31, you will need 5 0/1 variables. If the maximum value is 63, you will need 6 0/1 variables. In fact, if you use k 0/1 variables, the maximum value that can be represented is $2^k - 1$. You can write: $VMAX = 2^k - 1$. Taking logs, you can observe that the number of 0/1 variables required in this so-called binary expansion is approximately proportional to the log of the maximum value X can take on.

Although this substitution is valid, it should be avoided if possible. Most integer programming algorithms are not very efficient when applied to models containing this substitution.

11.2.2 Minimum Batch Size Constraints

When there are substantial economies of scale in undertaking an activity regardless of its level, many decision makers will specify a minimum “batch” size for the activity. For example, a large brokerage firm may require that, if you buy any bonds from the firm, you must buy at least 100. A zero/one variable can enforce this restriction as follows. Let:

- x = activity level to be determined (e.g., no. of bonds purchased),
- y = a zero/one variable = 1, if and only if $x > 0$,
- B = minimum batch size for x (e.g., 100), and
- U = known upper limit on the value of x .

The following two constraints enforce the minimum batch size condition:

$$\begin{aligned} x &\leq Uy \\ By &\leq x. \end{aligned}$$

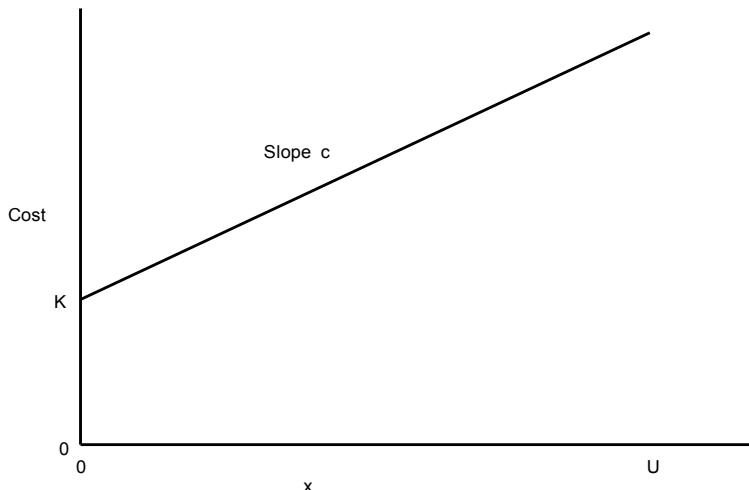
If $y = 0$, then the first constraint forces $x = 0$. While, if $y = 1$, the second constraint forces x to be at least B . Thus, y acts as a switch, which forces x to be either 0 or greater than B . The constant U should be chosen with care. For reasons of computational efficiency, it should be as small as validly possible.

Some IP packages allow the user to directly represent minimum batch size requirements by way of so-called semi-continuous variables. A variable x is semi-continuous if it is either 0 or in the range $B \leq x \leq \infty$. No binary variable need be explicitly introduced.

11.2.3 Fixed Charge Problems

A situation closely related to the minimum batch size situation is one where the cost function for an activity is of the fixed plus linear type indicated in Figure 11.1:

Figure 11.1 A Fixed Plus Linear Cost Curve



Define x , y , and U as before, and let K be the fixed cost incurred if $x > 0$. Then, the following components should appear in the formulation:

$$\begin{aligned} \text{Minimize} \quad & Ky + cx + \dots \\ \text{subject to} \quad & x \leq Uy \\ & \dots \\ & \dots \end{aligned}$$

The constraint and the term Ky in the objective imply x cannot be greater than 0 unless a cost K is incurred. Again, for computational efficiency, U should be as small as validly possible.

11.2.4 The Simple Plant Location Problem

The Simple Plant Location Problem (SPL) is a commonly encountered form of fixed charge problem. It is specified as follows:

- n = the number of sites at which a plant may be located or opened,
- m = the number of customer or demand points, each of which must be assigned to a plant,
- k = the number of plants which may be opened,
- f_i = the fixed cost (e.g., per year) of having a plant at site i , for $i = 1, 2, \dots, n$,
- c_{ij} = cost (e.g., per year) of assigning customer j to a plant at site i , for $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$.

The goal is to determine the set of sites at which plants should be located and which site should service each customer.

A situation giving rise to the SPL problem is the lockbox location problem encountered by a firm with customers scattered over a wide area. The plant sites, in this case, correspond to sites at which the firm might locate a postal lockbox that is managed by a bank at the site. The customer points would correspond to the, 100 say, largest metropolitan areas in the firm's market. A customer would mail his or her monthly payments to the closest lockbox. The reason for resorting to multiple lockboxes rather than having all payments mailed to a single site is several days of mail time may be saved. Suppose a firm receives \$60 million per year through the mail. The yearly cost of capital to the firm is 10% per year, and it could reduce the mail time by two days. This reduction has a yearly value of about \$30,000.

The f_i for a particular site would equal the yearly cost of having a lockbox at site i regardless of the volume processed through the site. The cost term c_{ij} would approximately equal the product: (daily cost of capital) \times (mail time in days between i and j) \times (yearly dollar volume mailed from area j).

Define the decision variables:

$$y_i = 1 \text{ if a plant is located at site } i, \text{ else } 0,$$

$$x_{ij} = 1 \text{ if the customer } j \text{ is assigned to a plant site } i, \text{ else } 0.$$

A compact formulation of this problem as an IP is:

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

$$\text{subject to } \sum_{i=1}^n x_{ij} = 1 \quad \text{for } j = 1 \text{ to } m, \quad (2)$$

$$\sum_{j=1}^m x_{ij} \leq my_i \quad \text{for } i = 1 \text{ to } n, \quad (3)$$

$$\sum_{i=1}^n y_i = k, \quad (4)$$

$$y_i = 0 \text{ or } 1 \quad \text{for } i = 1 \text{ to } n, \quad (5)$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for } i = 1 \text{ to } n, j = 1 \text{ to } m. \quad (6)$$

The constraints in (2) force each customer j to be assigned to exactly one site. The constraints in (3) force a plant to be located at site i if any customer is assigned to site i .

The reader should be cautioned against trying to solve a problem formulated in this fashion because the solution process may require embarrassingly much computer time for all, but the smallest problem. The difficulty arises because, when the problem is solved as an LP (i.e., with the conditions in (5) and (6) deleted), the solution tends to be highly fractional and with little similarity to the optimal IP solution.

A "tighter" formulation, which frequently produces an integer solution naturally when solved as an LP, is obtained by replacing (3) by the formula:

$$x_{ij} \leq y_i \text{ for } i = 1 \text{ to } n, j = 1 \text{ to } m. \quad (3')$$

At first glance, replacing (3) by (3') may seem counterproductive. If there are 20 possible plant sites and 60 customers, then the set (3) would contain 20 constraints, whereas set (3') would contain $20 \times 60 = 1,200$ constraints. Empirically, however, it appears to be the rule rather than the exception that, when the problem is solved as an LP with (3') rather than (3), the solution is naturally integer.

11.2.5 The Capacitated Plant Location Problem (CPL)

The CPL problem arises from the SPL problem if the volume of demand processed through a particular plant is an important consideration. In particular, the CPL problem assumes each customer has a known volume and each plant site has a known volume limit on total volume assigned to it. The additional parameters to be defined are:

D_j = volume or demand associated with customer j ,

K_i = capacity of a plant located at i

The IP formulation is:

$$\text{Minimize} \quad \sum_{i=1}^n f_i y_i + \sum_{i=1}^n \sum_{j=1}^m c_{ij} x_{ij} \quad (7)$$

$$\text{subject to} \quad \sum_{i=1}^n x_{ij} = 1 \quad \text{for } j = 1 \text{ to } m \quad (8)$$

$$\sum_{j=1}^m D_j x_{ij} \leq K_i y_i \quad \text{for } i = 1 \text{ to } n \quad (9)$$

$$x_{ij} \leq y_i \quad \text{for } i = 1 \text{ to } n, j = 1 \text{ to } m. \quad (10)$$

$$y_i = 0 \text{ or } 1 \quad \text{for } i = 1 \text{ to } n \quad (11)$$

$$x_{ij} = 0 \text{ or } 1 \quad \text{for } i = 1 \text{ to } n, j = 1 \text{ to } m. \quad (12)$$

This is the “single-sourcing” version of the problem. Because the variables x_{ij} are restricted to 0 or 1, each customer must have all of its volume assigned to a single plant. If “split-sourcing” is allowed, then the variables x_{ij} are allowed to be fractional with the interpretation that x_{ij} is the fraction of customer j ’s volume assigned to plant site i . In this case, condition (12) is dropped. Split sourcing, considered alone, is usually undesirable. An example is the assignment of elementary schools to high schools. Students who went to the same elementary school prefer to be assigned to the same high school.

Example: Capacitated Plant Location

Some of the points just mentioned will be illustrated with the following example.

The Zzyzx Company of Zzyzx, California currently has a warehouse in each of the following cities: (A) Baltimore, (B) Cheyenne, (C) Salt Lake City, (D) Memphis, and (E) Wichita. These warehouses supply customer regions throughout the U.S. It is convenient to aggregate customer areas and consider the customers to be located in the following cities: (1) Atlanta, (2) Boston, (3) Chicago, (4) Denver, (5) Omaha, and (6) Portland, Oregon. There is some feeling that Zzyzx is “overwarehoused”. That is, it may be able to save substantial fixed costs by closing some warehouses without unduly increasing transportation and service costs. Relevant data has been collected and assembled on a “per month” basis and is displayed below:

Cost per Ton-Month Matrix

Warehouse	Demand City						Monthly Supply Capacity in Tons	Monthly Fixed Cost
	1	2	3	4	5	6		
A	\$1675	\$400	\$685	\$1630	\$1160	\$2800	18	\$7,650
B	1460	1940	970	100	495	1200	24	3,500
C	1925	2400	1425	500	950	800	27	3,500
D	380	1355	543	1045	665	2321	22	4,100
E	922	1646	700	508	311	1797	31	2,200
Monthly Demand in Tons	10	8	12	6	7	11		

For example, closing the warehouse at A (Baltimore) would result in a monthly fixed cost saving of \$7,650. If 5 (Omaha) gets all of its monthly demand from E (Wichita), then the associated transportation cost for supplying Omaha is $7 \times 311 = \$2,177$ per month. A customer need not get all of its supply from a single source. Such “multiple sourcing” may result from the limited capacity of each warehouse (e.g., Cheyenne can only process 24 tons per month. Should Zzyzx close any warehouses and, if so, which ones?)

We will compare the performance of four different methods for solving, or approximately solving, this problem:

- 1) Loose formulation of the IP.
- 2) Tight formulation of the IP.
- 3) Greedy open heuristic: start with no plants open and sequentially open the plant giving the greatest reduction in cost until it is worthless to open further plants.
- 4) Greedy close heuristic: start with all plants open and sequentially close the plant saving the most money until it is worthless to close further plants.

The advantage of heuristics 3 and 4 is they are easy to apply. The performance of the four methods is as follows:

Method	Objective value: Best Solution	Computing Time in Seconds	Plants Open	Objective value: LP Solution
Loose IP	46,031	3.38	A,B,D	35,662
Tight IP	46,031	1.67	A,B,D	46,031
Greedy Open Heuristic	46,943	nil	A,B,D,E	—
Greedy Close Heuristic	46,443	nil	A,C,D,E	—

Notice, even though the loose IP finds the same optimum as the tight formulation (as it must), it takes about twice as much computing time. For large problems, the difference becomes much more dramatic. Notice for the tight formulation, however, the objective function value for the LP solution is the same as for the IP solution. When the tight formulation was *solved as an LP*, the solution was naturally integer.

The single product dynamic lotsizing problem is described by the following parameters:

- n = number of periods for which production is to be planned for a product;
- D_j = predicted demand in period j , for $j = 1, 2, \dots, n$;
- f_i = fixed cost of making a production run in period i ;
- h_i = cost per unit of product carried from period i to $i + 1$.

This problem can be cast as a simple plant location problem if we define:

$$c_{ij} = D_j \sum_{t=i}^{j-1} h_t$$

That is, c_{ij} is the cost of supplying period j 's demand from period i production. Each period can be thought of as both a potential plant site (period for a production run) and a customer.

If, further, there is a finite production capacity, K_i , in period i , then this capacitated dynamic lotsizing problem is a special case of the capacitated plant location problem.

Dual Prices and Reduced Costs in Integer Programs

Dual prices and reduced costs in solution reports for integer programs have a restricted interpretation. For first time users of IP, it is best to simply disregard the reduced cost and dual price column in the solution report. For the more curious, the dual prices and reduced costs in a solution report are obtained from the linear program that remains after all integer variables have been fixed at their optimal values and removed from the model. Thus, for a pure integer program (i.e., all variables are required to be integer), you will generally find:

- all dual prices are zero, and
- the reduced cost of a variable is simply its objective function coefficient (with sign reversed if the objective is MAX).

For mixed integer programs, the dual prices may be of interest. For example, for a plant location problem where the location variables are required to be integer, but the quantity-shipped variables are continuous, the dual prices reported are those from the continuous problem where the locations of plants have been specified beforehand (at the optimal locations).

11.2.6 Modeling Alternatives with the Scenario Approach

We may be confronted by alternatives in two different ways: a) we have to choose among two or more alternatives and we want to figure out which is best, or b) nature or the market place will choose one of two or more alternatives, and we are not sure which alternative nature will choose, so we want to analyze all alternatives so we will be prepared to react optimally once we learn which alternative was chosen by nature. Here we consider only situation (a). We call the approach the scenario approach or the disjunctive formulation, see for example Balas(1979) or section 16.2.3 of Martin(1999).

Suppose that if we disregard the alternatives, our variables are simply called x_1, x_2, \dots, x_n . We call the conditions that must hold if alternative s is chosen, scenario s . Without too much loss of generality, we assume all variables are non-negative. The scenario/disjunctive approach to formulating a discrete decision problem proceeds as follows:

For each scenario s :

- 1) Write all the constraints that must hold if scenario s is chosen.
- 2) For all variables in these constraints add a subscript s , to distinguish them from equivalent variables in other scenarios. So x_j in scenario s becomes x_{sj} .
- 3) Add a 0/1 variable, y_s , to the model with the interpretation that $y_s = 1$ if scenario s is chosen, else 0.
- 4) Multiply the RHS constant term of each constraint in scenario s by y_s .
- 5) For each variable x_{sj} that appears in any of the scenario s constraints, add the constraint:

$x_{sj} \leq M^* y_s$, where M is a large positive constant. The purpose of this step is to force all variables in scenario s to be 0 if scenario s is not chosen.

Finally, we tie all the scenarios together with:

$$\sum_s y_s = 1, \text{ i.e., we must choose one scenario;}$$

For each variable x_j , add the constraint:

$$x_j = \sum_s x_{sj}, \text{ so } x_j \text{ takes on the value appropriate to which scenario was chosen.}$$

For example, if just after step 1 we had a constraint of the form:

$$\sum_j a_{sj} * x_j \leq a_{s0},$$

then steps 2-4 would convert it to:

$$\sum_j a_{sj} * x_{sj} \leq a_{s0} * y_s,$$

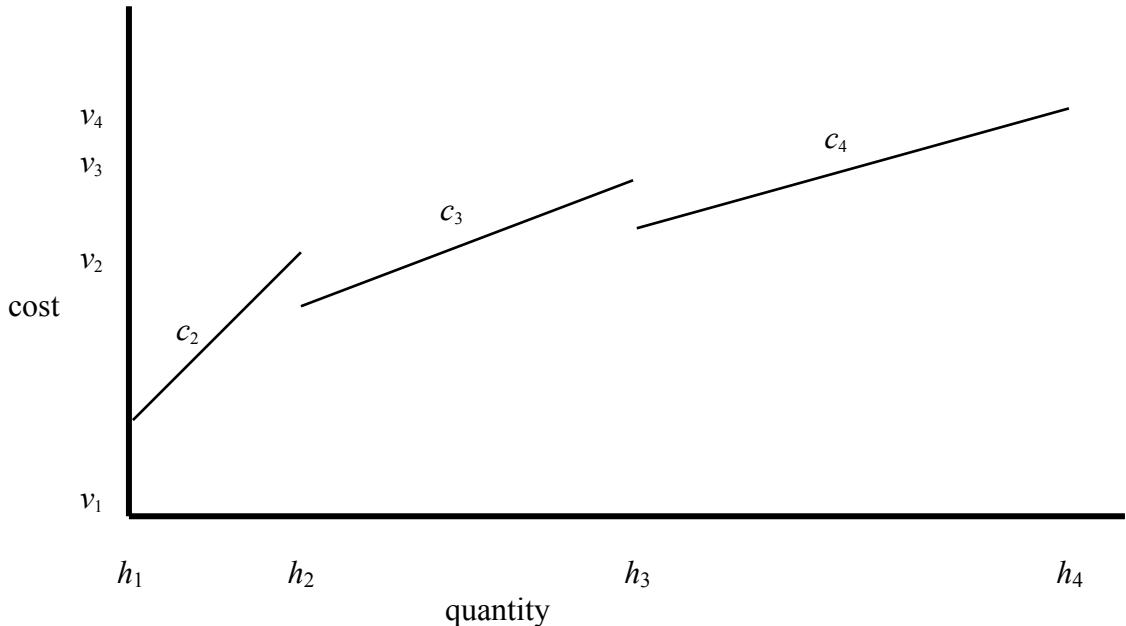
The forcing constraints in step 5 are not needed if $y_s = 0$ implies $x_{sj} = 0$, e.g., if all the a_{sj} are nonnegative and the x_j are constrained to be nonnegative.

A somewhat similar approach to the disjunctive/scenario approach is the RLT approach developed by Adams and Sherali(2005). The next section illustrates the scenario approach for representing a decision problem.

11.2.7 Linearizing a Piecewise Linear Function, Discontinuous Case

If you ask a vendor to provide a quote for selling you some quantity of material, the vendor will typically offer a quantity discount function that looks something like that shown in Figure 11.2

Figure 11.2 Quantity Discount Piecewise Linear Discontinuous Cost Curve



Define:

c_s = slope of piecewise linear segment s ,
 h_s, v_s = horizontal and vertical coordinates of the rightmost point of segment s .

Note that segment 1 is the degenerate segment of buying nothing. This example illustrates that we do not require that a piecewise linear function be continuous.

Let us consider the following situation:

We pay \$50 if we buy anything in a period, plus
\$2.00/unit if quantity < 100 ,
\$1.90/unit if quantity ≥ 100 but < 1000 ,
\$1.80/unit if ≤ 1000 but ≤ 5000 .

We assume h_s, v_s, c_s are constants, and $h_s \leq h_{s+1}$. It then follows that:

h	v	$c =$
0	0	0
100	250	2
1000	1950	1.90
5000	9050	1.80;

We will describe two ways of representing piecewise linear functions: first the disjunctive method, and then the convex weighting or lambda method. Let x denote the amount we decide to purchase. Using step 1 of the scenario or disjunctive formulation approach,

if segment/scenario 1 is chosen, then

$$cost = 0;$$

$$x = 0;$$

If segment/scenario 2 is chosen, then

$$cost = v_2 - c_2 * (h_2 - x); \quad [\text{or } 250 - 2*(100 - x)],$$

$$x \leq h_2; \quad [\text{or } x \leq 100],$$

$$x \geq h_1; \quad [\text{or } x \geq 0],$$

Similar constraints apply for scenario/segments 3 and 4. We assume that fractional values, such as $x = 99.44$ are allowed, else we would write $x \leq 99$ rather than $x \leq 100$ above.

If we apply steps 2-4 of the scenario formulation method, then we get:

For segment/scenario 1 is chosen, then

$$cost_1 = 0;$$

$$x_1 = 0;$$

If segment/scenario 2 is chosen, then

$$cost_2 = v_2 * y_2 - c_2 * h_2 * y_2 + c_2 * x_2; \quad [\text{or } cost_2 = 50 * y_2 + 2 * x_2],$$

$$x_2 \leq h_2 * y_2; \quad [\text{or } x_2 \leq 100 * y_2],$$

$$x_2 \geq h_1 * y_2; \quad [\text{or } x_2 \geq 0 * y_2],$$

If segment/scenario 3 is chosen, then

$$cost_3 = v_3 * y_3 - c_3 * h_3 * y_3 + c_3 * x_3; \quad [\text{or } cost_3 = 50 * y_3 + 1.9 * x_3],$$

$$x_3 \leq h_3 * y_3; \quad [\text{or } x_3 \leq 1000 * y_3],$$

$$x_3 \geq h_2 * y_3; \quad [\text{or } x_3 \geq 100 * y_3],$$

If segment/scenario 4 is chosen, then

$$cost_4 = v_4 * y_4 - c_4 * h_4 * y_4 + c_4 * x_4; \quad [\text{or } cost_4 = 50 * y_4 + 1.8 * x_4]$$

$$x_4 \leq h_4 * y_4; \quad [\text{or } x_4 \leq 5000 * y_4],$$

$$x_4 \geq h_3 * y_4; \quad [\text{or } x_4 \geq 1000 * y_4],$$

We must choose one of the four segments, so:

$$y_1 + y_2 + y_3 + y_4 = 1;$$

$$y_1, y_2, y_3, y_4 = 0 \text{ or } 1;$$

and the true quantity and cost are found with:

$$x_1 + x_2 + x_3 + x_4 = x;$$

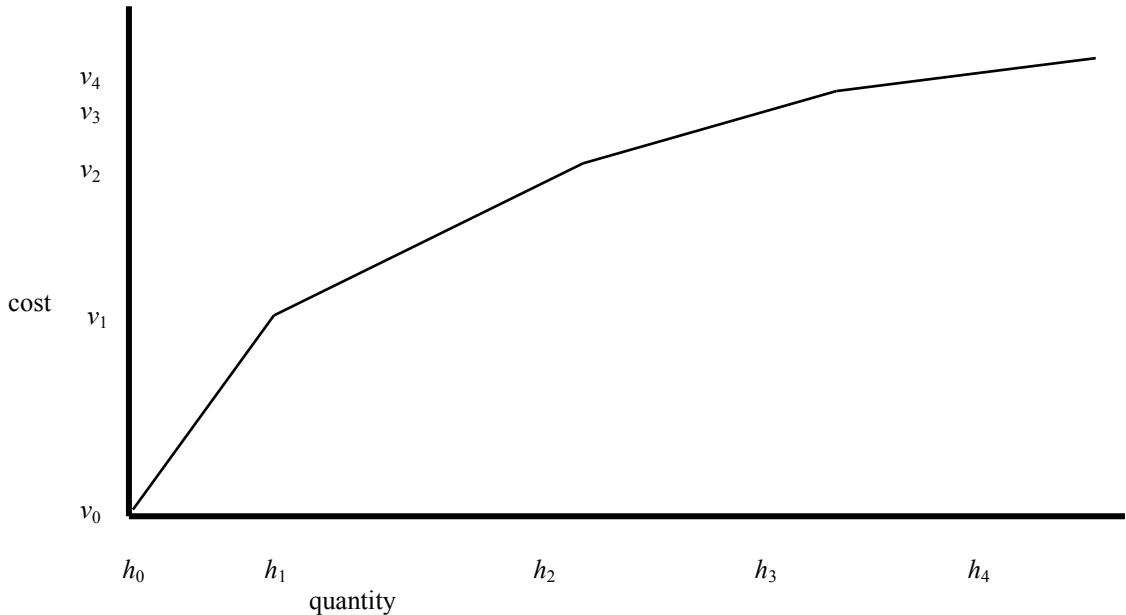
$$cost_1 + cost_2 + cost_3 + cost_4 = cost;$$

11.2.8 Linearizing a Piecewise Linear Function, Continuous Case

The previous quantity discount example illustrated what is called an “all units discount”. Sometimes, a vendor will instead quote an incremental units discount, in which the discount applies only to the units above a threshold. The following example illustrates. The first 1,000 liters of the product can be

purchased for \$2 per liter. The price drops to \$1.90 per liter for units beyond 1000, \$1.80 for units above 3500, and \$1.75 for units beyond 5000. At most 7000 liters can be purchased.

Figure 11.3 Continuous Piecewise Linear Cost Curve



Verify that the corresponding values for the h_i and v_i are:

i	h	v
0	0	0
1	1000	2000
2	3500	6750
3	5000	9450
4	7000	12950

Such continuous piecewise linear functions are found not only in purchasing but also are frequently used in the modeling of energy conversion processes such as the generation of electricity. The amount of electrical energy produced by a hydro-electric or fossil fuel burning generator may be a nonlinear function of the input volume of water or fuel.

Define the variables:

w_i = nonnegative weight to be applied to point i , for $i = 0, 1, 2, 3, 4$.
 x = amount purchased,
 $cost$ = total cost of the purchase.

We can cause x and $cost$ to almost be calculated correctly by writing the constraints:

$$\begin{aligned}x &= w_0h_0 + w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4; \\cost &= w_0v_0 + w_1v_1 + w_2v_2 + w_3v_3 + w_4v_4; \\1 &= w_0 + w_1 + w_2 + w_3 + w_4;\end{aligned}$$

Any point on the line segment connecting the two points (h_i, v_i) and (h_{i+1}, v_{i+1}) can be represented by choosing appropriate values for w_i and w_{i+1} so that $w_i + w_{i+1} = 1$, and $w_i, w_{i+1} \geq 0$. This method is sometimes called the lambda method because the Greek symbol lambda was used originally to represent the weights. To ensure that the point corresponding to a particular set of values for the w_i lies on the curve, we need to require that if two or more of the w_i are > 0 , they must be adjacent. We said “almost” in the earlier sentence because there is nothing in the three constraints above that enforce this adjacency condition. There are two ways of enforcing this adjacency condition: a) declare the w_i to be members of an SOS2 set in LINGO, or b) add a number of binary variables to enforce the condition.

The following code fragment illustrates how to use the SOS2 feature in LINGO.

```
! Representing a continuous piecewise linear
function in LINGO using the SOS2 feature;

x = w0*0 + w1*1000 + w2*3500 + w3*5000 + w4* 7000;
cost = w0*0 + w1*2000 + w2*6750 + w3*9450 + w4*12950;
1 = w0 + w1 + w2 + w3 + w4;

! The ordering/adjacency of the variables in the SOS2 set
is determined by the order of declarations. The SOS2 feature
restricts the number of nonzero values in the set to be at
most 2, and if 2, they must be adjacent;
@SOS2('MySOS2',w0); @SOS2('MySOS2',w1); @SOS2('MySOS2',w2);
@SOS2('MySOS2',w3); @SOS2('MySOS2',w4);
```

If you arbitrarily add the constraint, $X = 6000$, and solve, you get the solution:

Variable	Value
X	6000.000
W0	0.000000
W1	0.000000
W2	0.000000
W3	0.5000000
W4	0.5000000
COST	11200.00

If for some reason you do not want to use the SOS2 feature, you can introduce 4 binary variables:
 $y_i = 1$ if x is in the interval with endpoints h_{i-1} and h_i , for $i = 1, 2, 3, 4$. We would replace the SOS2 declarations by the constraints:

```
! The y's must be binary;
@BIN(y1); @BIN(y2); @BIN(y3); @BIN(y4);
```

```

! Some interval must be chosen;
y1 + y2 + y3 + y4 = 1;
! If point i has any weight, then one of the adjacent
intervals must be chosen;
w0 <= y1;
w1 <= y1 + y2;
w2 <= y2 + y3;
w3 <= y3 + y4;
w4 <= y4;

```

11.2.9 An n Interval Piecewise Linear Function Using $\log(n)$ Binaries

The previous example used n binary variables to enforce the choosing of one alternative out of n . With a little ingenuity this “choose one out of n ” requirement can be enforced with only order of $\log_2(n)$ binary variables. We illustrate for the case of eight intervals, for which we need three binary variables, y_1, y_2, y_3 . Denote the 8 intervals by 0, 1, ..., 7, with point v_i being the left boundary of interval i . We will assign binary variables to intervals thus:

If the interval is one of 4, 5, 6, 7, then $y_3 = 1$, else 0,

If the interval is one of 2, 3, 4, 5, then $y_2 = 1$, else 0,

If the interval is one of 1, 2, 5, 6, then $y_1 = 1$, else 0;

Thus, we also need the constraints:

$$\begin{aligned} w_0 + w_1 + w_2 + w_3 &\leq 1 - y_3; \\ w_5 + w_6 + w_7 + w_8 &\leq y_3; \\ w_0 + w_1 + w_7 + w_8 &\leq 1 - y_2; \\ w_3 + w_4 + w_5 &\leq y_2; \\ w_0 + w_4 + w_8 &\leq 1 - y_1; \\ w_2 + w_6 &\leq y_1; \\ y_1, y_2, y_3 &= 0 \text{ or } 1; \end{aligned}$$

Notice that if:

$y_1 = y_2 = y_3 = 0$, then $w_2 + w_3 + w_4 + w_5 + w_6 + w_7 + w_8 = 0$, i.e., the interval is 0,

$y_1 = 1, y_2 = y_3 = 0$, then $w_0 + w_3 + w_4 + w_5 + w_6 + w_7 + w_8 = 0$, i.e., the interval is 1, etc.

It may be of interest to note that this is a “Gray” binary coding of 0, 1, ..., 7, in that exactly 1 “bit” of y_1, y_2, y_3 changes in the binary representation as one moves from i to $i+1$.

Piecewise Linear Approximations to Multivariate Functions

Suppose we have a function of two variables:

$$cost = f(x, y).$$

We can construct a piecewise linear approximation to this function if we

choose n points, (x_{bar_i}, y_{bar_i}) for $i = 1, 2, \dots, n$,

e.g., corner points of the triangles in Figure 11.4, and,

introduce the n nonnegative variables, w_i , and

add the constraints:

$$\sum_i w_i = 1,$$

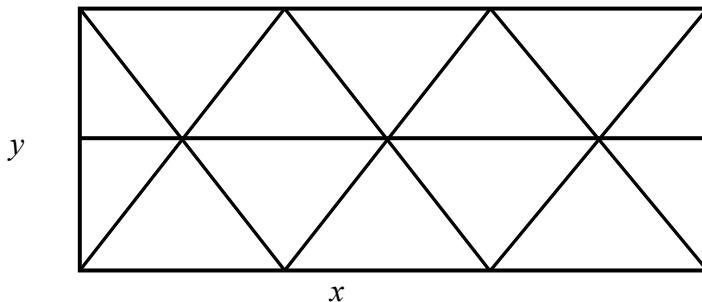
$$cost = \sum_i w_i f(x_{bar_i}, y_{bar_i}),$$

$$x = \sum_i w_i x_{bar_i},$$

$$y = \sum_i w_i ybar_i,$$

If we are lucky, e.g., $f(x,y)$ is convex in the appropriate way then: a) at most three of the w_i will be nonzero, and b) the nonzero w_i will correspond to adjacent points, that is, corner points of a triangle containing no other points.

Figure 11.4 Triangulation of x,y Space



If we are unlucky, then we have to introduce 0/1 variables. If we are lazy and are willing to restrict the solution to one of the n sampled points, then all we have to do is declare the w_i variables to be 0/1.

If we want to allow any possible combination of x and y , then we have to make sure the n points describe a triangulation of the x,y space, as in Figure 11.4, introduce a 0/1 variable z_i for each triangle, and then force exactly one triangle to be chosen with the constraint:

$$\sum_i z_i = 1;$$

We must also add constraints that say that if any weight is applied to point i , then the chosen (x,y) must be in one of the triangles for which point i is a corner. More formally:

$$w_i \leq \sum_{j \in T(i)} z_j, \text{ for each point } i, \text{ where } T(i) \text{ is the set of triangles (there should be at most 6) for which point } i \text{ is a corner point.}$$

11.2.10 Converting Multivariate Functions to Separable Functions

The previous methods are applicable only to piecewise linear functions. There are some standard methods available for transforming a certain functions of several variables, so a function is obtained that is additively separable in the transformed variables. The most common such transformation is for converting a product of two variables into separable form. For example, given the function:

$$x_1 * x_2,$$

add the linear constraints:

$$y_1 = (x_1 + x_2)/2$$

$$y_2 = (x_1 - x_2)/2.$$

Then, replace every instance of $x_1 * x_2$ by the term $y_1^2 - y_2^2$. That is, the claim is:

$$x_1 * x_2 = y_1^2 - y_2^2.$$

The justification is observed by noting:

$$\begin{aligned} y_1^2 - y_2^2 &= (x_1^2 + 2 * x_1 * x_2 + x_2^2)/4 \\ &- (x_1^2 - 2 * x_1 * x_2 + x_2^2)/4 \\ &= 4 * x_1 * x_2 / 4 = x_1 * x_2 \end{aligned}$$

This example suggests that, any time you have a product of two variables, you can add two new variables to the model and replace the product term by a sum of two squared variables. If you have n original variables, you could have up to $n(n-1)/2$ cross product terms. This suggests that you might need up to $n(n-1)$ new variables to get rid of all cross product terms. In fact, the above ideas can be generalized, using various factorization techniques such as Cholesky and others, so only n new variables are needed.

11.3 Outline of Integer Programming Methods

The time a computer requires to solve an IP may depend dramatically on how you formulated it. It is, therefore, worthwhile to know a little about how IPs are solved. There are two general approaches for solving IPs: “cutting plane” methods and “branch-and-bound” (B & B) method. For a comprehensive introduction to integer programming solution methods, see Nemhauser and Wolsey (1988), and Wolsey (1998). Most commercial IP programs use the B & B method, but aided by some cutting plane features. We will first describe the B & B method. In most general terms, B & B is a form of intelligent enumeration.

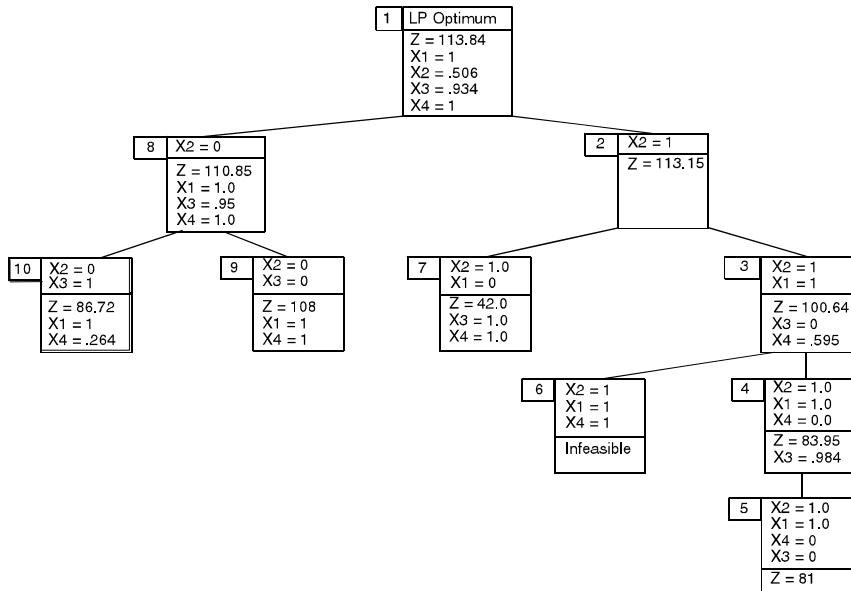
More specifically, B & B first solves the problem as an LP. If the LP solution is integer valued in the integer variables, then no more work is required. Otherwise, B & B resorts to an intelligent search of all possible ways of rounding the fractional variables.

We shall illustrate the application of the branch-and-bound method with the following problem:

```
MAX= 75 * X1 + 6 * X2 + 3 * X3 + 33 * X4;
    774 * X1 + 76 * X2 + 22 * X3 + 42 * X4 <= 875;
    67 * X1 + 27 * X2 + 794 * X3 + 53 * X4 <= 875;
@BIN( X1); @BIN( X2); @BIN( X3); @BIN( X4);
```

The search process a computer might follow in finding an integer optimum is illustrated in Figure 11.5. First, the problem is solved as an LP with the constraints $X1, X2, X3, X4 \leq 1$. This solution is summarized in the box labeled 1. The solution has fractional values for $X2$ and $X3$ and is, therefore, unacceptable. At this point, $X2$ is arbitrarily selected and the following reasoning is applied. At the integer optimum, $X2$ must equal either 0 or 1.

Figure 11.5 Branch-and-Bound Search Tree



Therefore, replace the original problem by two new subproblems. One with X_2 constrained to equal 1 (box or node 2) and the other with X_2 constrained to equal 0 (node 8). If we solve both of these new IPs, then the better solution must be the best solution to the original problem. This reasoning is the motivation for using the term “branch”. Each subproblem created corresponds to a branch in an enumeration tree.

The numbers to the upper left of each node indicate the order in which the nodes (or equivalently, subproblems) are examined. The variable Z is the objective function value. When the subproblem with X_2 constrained to 1 (node 2) is solved as an LP, we find X_1 and X_3 take fractional values. If we argue as before, but now with variable X_1 , two new subproblems are created:

Node 7) one with X_1 constrained to 0 , and

Node 3) one with X_1 constrained to 1.

This process is repeated with X_4 and X_3 until node 5. At this point, an integer solution with $Z = 81$ is found. We do not know this is the optimum integer solution, however, because we must still look at subproblems 6 through 10. Subproblem 6 need not be pursued further because there are no feasible solutions having all of X_2 , X_1 , and X_4 equal to 1. Subproblem 7 need not be pursued further because it has a Z of 42, which is worse than an integer solution already in hand.

At node 9, a new and better integer solution with $Z = 108$ is found when $X3$ is set to 0. Node 10 illustrates the source for the “bound” part of “branch-and-bound”. The solution is fractional. However, it is not examined further because the Z -value of 86.72 is less than the 108 associated with an integer solution already in hand. The Z -value at any node is a bound on the Z -value at any offspring node. This is true because an offspring node or subproblem is obtained by appending a constraint to the parent problem. Appending a constraint can only hurt. Interpreted in another light, this means the Z -values cannot improve as one moves down the tree. The tree presented in the preceding figure was only one illustration of how the tree might be searched. Other trees could be developed for the same problem by playing with the following two degrees of freedom:

- (a) Choice of next node to examine, and
- (b) Choice of how the chosen node is split into two or more subnodes.

For example, if nodes 8 and then 9 were examined immediately after node 1, then the solution with $Z = 108$ would have been found quickly. Further, nodes 4, 5, and 6 could then have been skipped because the Z -value at node 3 (100.64) is worse than a known integer solution (108), and, therefore, no offspring of node 3 would need examination.

In the example tree, the first node is split by branching on the possible values for $X2$. One could have just as well chosen $X3$ or even $X1$ as the first branching variable.

The efficiency of the search is closely related to how wisely the choices are made in (a) and (b) above. Typically, in (b) the split is made by branching on a single variable. For example, if, in the continuous solution, $x = 1.6$, then the obvious split is to create two subproblems. One with the constraint $x \leq 1$, and the other with the constraint $x \geq 2$. The split need not be made on a single variable. It could be based on an arbitrary constraint. For example, the first subproblem might be based on the constraint $x_1 + x_2 + x_3 \leq 0$, while the second is obtained by appending the constraint $x_1 + x_2 + x_3 \geq 1$. Also, the split need not be binary. For example, if the model contains the constraint $y_1 + y_2 + y_3 = 1$, then one could create three subproblems corresponding to either $y_1 = 1$, or $y_2 = 1$, or $y_3 = 1$.

If the split is based on a single variable, then one wants to choose variables that are “decisive.” In general, the computer will make intelligent choices and the user need not be aware of the details of the search process. The user should, however, keep the general B & B process in mind when formulating a model. If the user has *a priori* knowledge that an integer variable x is decisive, then for the LINGO program it is useful to place x early in the formulation to indicate its importance. This general understanding should drive home the importance of a “tight” LP formulation. A tight LP formulation is one which, when solved, has an objective function value close to the IP optimum. The LP solutions at the subproblems are used as bounds to curtail the search. If the bounds are poor, many early nodes in the tree may be explicitly examined because their bounds look good even though, in fact, these nodes have no good offspring.

Cutting planes are very important for solving certain classes of IP’s. Some of these difficult IP’s would take prohibitively long to solve with just B&B, without the use of cutting planes. A cutting plane is an additional constraint that is added to the formulation to remove fractional points from the LP relaxation. Thus, if some good cuts have been added, the chance is much higher that when the LP relaxation is solved, a much higher fraction of the integer variables will take on naturally integer values.

There is a wide variety of cuts that are implemented in commercial IP solvers, an even wider variety of cuts that have been described in the optimization literature. One of the most general types of cuts is the Mixed Integer Rounding, or MIR, cut. A very similar cut described in the literature is the

Gomory Mixed Integer cut. We will illustrate the MIR cut with a little example. Suppose we want to solve the little mixed integer program:

```
MIN = 5*y + 3*u + 4*v;
     8*y + u - v = 13;
@GIN( y)
```

As usual, by default, all variables are restricted to be ≥ 0 . If you delete the requirement that y be integer and solve the resulting LP, you get the fractional solution $y = 1.625$, $u = v = 0$. Now we reason that in any feasible integer solution, either :

Case 1: $y \leq 1$ and $u \geq 5$, or

Case 2: $y \geq 2$ and $v \geq 3$.

So in any integer feasible solution, we must have either:

$u \geq 5$, or

$v \geq 3$.

Multiplying by either 3 or 5, we must have either:

$3*u \geq 3*5$, or

$5*v \geq 3*5$.

Because $v, u \geq 0$, we can add $5*v$ to the first constraint, and $3*u$ to the second constraint without destroying their validity, so we must have either:

$3*u + 5*v \geq 3*5$, or

$3*u + 5*v \geq 3*5$,

so the single constraint or cut is justified:

$3*u + 5*v \geq 3*5$.

Notice that this cut cuts off the fractional solution $y = 1.625$, $u = v = 0$. With this cut added, when we solve the LP:

```
MIN = 5*y + 3*u + 4*v;
     8*y + u - v = 13;
     3*u + 5*v >= 15;
```

We get the naturally integer solution;

```
Global optimal solution found.
Objective value:          20.000000
```

Variable	Value
Y	1.000000
U	5.000000
V	0.000000

With a little bit of imagination, e.g., by replacing y by a sum of integer variables with integer coefficients, and replacing u and v by positive weighted sums of nonnegative variables, a wide variety of MIR type cuts are possible.

11.4 Computational Difficulty of Integer Programs

Integer programs can be very difficult to solve. This is in marked contrast to LP problems. The solution time for an LP is fairly predictable. For an LP, the time increases approximately proportionally with the number of variables and approximately with the square of the number of constraints. For a given IP problem, the time may in fact decrease as the number of constraints is

increased. As the number of integer variables is increased, the solution time may increase dramatically. Some small IPs (e.g., 6 constraints, 60 variables) are extremely difficult to solve.

Just as with LPs, there may be alternate IP formulations of a given problem. With IPs, however, the solution time generally depends critically upon the formulation. Producing a good IP formulation requires skill. For many of the problems in the remainder of this chapter, the difference between a good formulation and a poor formulation may be the difference between whether the problem is solvable or not.

11.4.1 NP-Complete Problems

Integer programs belong to a class of problems known as *NP*-hard. We may somewhat loosely think of *NP* as meaning "not polynomial". This means that there is no known algorithm of solving these problems such that the computational effort at worst increases as a polynomial in the problem size. For our purposes, we will say that the computational complexity of an algorithm is polynomial if there is a positive constant k , such that the time to solve a problem of size n is proportional to n^k . For example, sorting a set of n numbers can easily be done in (polynomial) time proportional to n^2 , ($n \log(n)$ if one is careful), whereas solving an integer program in n zero/one variables may, in the worst case, take (exponential) time proportional to 2^n . There may be a faster way, but no one has published an algorithm for integer programs that is guaranteed to take polynomial time on every problem presented to it. The terms *NP*-complete and *P*-complete apply to problems that can be stated as "yes/no" or feasibility problems. The yes/no variation of an optimization problem would be a problem of the form: Is there a feasible solution to this problem with cost less-than-or-equal-to 1250. In an optimization problem, we want a feasible solution with minimum cost. Khachian (1979) showed that the feasibility version of LP is solvable in polynomial time. So, we say LP is in *P*. Integer programming stated in feasibility form, and a wide range of similar problems, belong to a class of problems called *NP*-complete. These problems have the feature that it is possible to convert any one of these problems into any other *NP*-complete problem in time that is polynomial in the problem size. Thus, if we can convert problem *A* into problem *B* in polynomial time, then solve *B* in polynomial time, and then convert the solution to *B* to a valid solution to *A* in polynomial time, we then have a way of solving *A* in polynomial time.

The notable thing about *NP*-complete problems is that, if someone develops a guaranteed fast (e.g., polynomial worst case) time method for solving one of these problems, then that someone also has a polynomial time algorithm for every other *NP*-complete problem. An important point to remember is that the *NP*-completeness classification is defined in terms of worst case behavior, not average case behavior. For practical purposes, one is interested mainly in average case behavior. The current situation is that the average time to solve many important practical integer programming problems is quite short. The fact that someone may occasionally present us with an extremely difficult integer programming problem does not prevent us from profiting from the fact that a large number of practical integer programs can be solved rapidly. Perhaps the biggest open problem in modern mathematics is whether the problems in the *NP*-complete class are inherently difficult. This question is cryptically phrased as is: *P* = *NP*? Are these problems really difficult, or is it that we are just not smart enough to discover the universally fast algorithm? In fact, a "Millennium prize" of \$1,000,000 is offered by the Clay Mathematics Institute, www.claymath.org, for an answer to this question. For a more comprehensive discussion of the *NP*-complete classification, see Martin (1999).

11.5 Problems with Naturally Integer Solutions and the Prayer Algorithm

The solution algorithms for IP are generally based on first solving the IP as an LP by disregarding the integrality requirements and praying the solution is naturally integer. For example, if x is required to be 0 or 1, the problem is first solved by replacing this requirement by the requirement that simply $0 \leq x \leq 1$. When initiating the analysis of a problem in which integer answers are important, it is useful to know beforehand whether the resulting IP will be easy to solve. After the fact, one generally observes the IP was easy to solve if the objective function values for the LP optimum and the IP optimum were close. About the only way we can predict beforehand the objective function values of the LP and IP will be close is if we know beforehand the LP solution will be almost completely integer valued. Thus, we are interested in knowing what kinds of LPs have naturally integer solutions.

The classes of LP problems for which we know beforehand there is a naturally integer optimum have integer right-hand sides and are in one of the classes:

- (a) Network LPs,
- (b) MRP or Integral Leontief LPs,
- (c) Problems that can be transformed to (a) or (b) by either row operations or taking the dual.

We first review the distinguishing features of network and MRP LPs.

11.5.1 Network LPs Revisited

A LP is said to be a network LP if: 1) disregarding simple upper and lower bound constraints (such as $x \leq 3$), each variable appears in at most two constraints, and 2) if each variable appears in two constraints, its coefficients in the two are +1 and -1. If the variable appears in one constraint, its coefficient is either +1 or -1.

Result: If the right-hand side is integer, then there is an integer optimum. If the objective coefficients are all integer, then there is an optimum with integral dual prices.

11.5.2 Integral Leontief Constraints

A constraint set is said to be *integral Leontief* or MRP (for Material Requirements Planning) if (see Jeroslow, Martin, et al. (1992)):

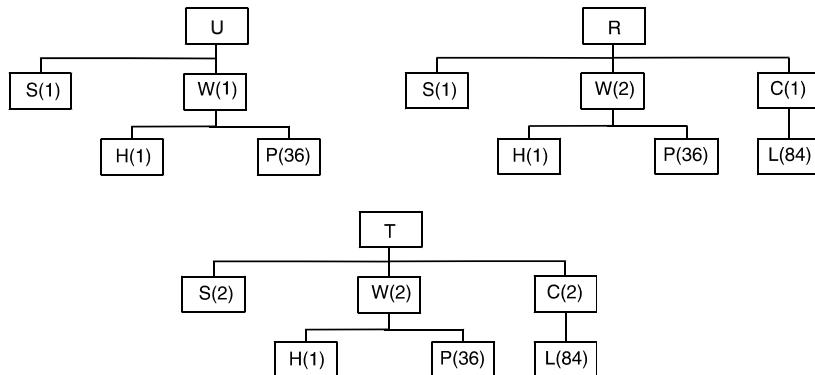
- Each constraint is an equality,
- Every column has exactly one positive coefficient and it is a +1,
- Each column has 0 or more negative coefficients, every one of which is integer,
- Each RHS coefficient is a nonnegative integer.

Result: An LP whose complete constraint set is an MRP set has an optimal solution that is integer. Further, if the objective coefficients are all integer, then there is an optimal solution with integral dual prices.

11.5.3 Example: A One-Period MRP Problem

The Schwindle Cycle Company makes three products: Unicycles (U), Regular Bicycles (R), and Twinbikes (T). Each product is assembled from a variety of components including: seats (S), wheels (W), hubs (H), spokes (P), chains (C), and links (L). The full bills of materials for each product are shown below. The numbers in parentheses specify how many units of the child are required per parent:

Figure 11.6 MRP Structure for Bicycles



Current inventories are zero. Schwindle needs to supply 100 Unicycles, 500 Regular bicycles, and 200 Twinbikes. Finished products and complete sub-assemblies can be either manufactured or bought at the following prices:

Item:	U	R	T	S	W	C	H	P	L
Bought Price:	2.60	5.2	3.10	0.25	1.40	0.96	0.19	0.07	0.05
Assembly Cost:	1.04	1.16	1.90	0.20	0.22	0.26	0.16	0.04	0.03

Note the assembly cost is the immediate cost at the level of assembly. It does not include the cost of the components going into the assembly. How many units of each item should be made or bought to satisfy demand at minimum price?

An LP formulation is:

```

MODEL:
SETS:
  TYPES/U, R, T/:M, B, MP, BP, NEED;
  MATERIALS/S, W, C/:MM, MB, MMP, MBP;
  SUBMATS/H, P, L/:SMM, SMB, SMP, SBP;
  REQ(TYPES, MATERIALS) : MATREQ;
  MREQ(MATERIALS, SUBMATS) : SMATREQ;
ENDSETS
DATA:
  NEED      = 100  500  200;
  MP        = 1.04 1.16  1.9;
  BP        = 2.6   5.2   3.1;
  MMP       = .2    .22   .26;
  MBP       = .25   1.4   .96;
  SMP       = .16   .04   .03;
  SBP       = .19   .07   .05;
  MATREQ   =
            1     1     0
            1     2     1
            2     2     2;
  SMATREQ =
            0     0     0
            1     36    0
            0     0     84;
ENDDATA
  MIN = @SUM(TYPES : M * MP + B * BP)
        + @SUM(MATERIALS : MM * MMP + MB * MBP)
        + @SUM(SUBMATS: SMM * SMP + SMB * SBP);
  @FOR(TYPES: M + B = NEED);
  @FOR(MATERIALS(I) : MM(I) + MB(I) =
    @SUM(TYPES(J) : M(J) * MATREQ(J, I)));
  @FOR(SUBMATS(I) : SMM(I) + SMB(I) =
    @SUM(MATERIALS(J) : MM(J) * SMATREQ(J, I)));
END

```

In the PICTURE of the formulation below, notice it has the MRP structure:

	U	U	R	R	T	T	S	S	W	W	C	C	H	H	P	P	L	L
	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B
1: A	A	A	A	A	A	T	T	T	A	T	T	T	T	T	U	U	U	U
UNICYCLE:	1	1	'	'	'	'	'	'	'	'	'	'	'	'	'	'	'	'
REGULAR:	'	1	1	'	'	'	'	'	'	'	'	'	'	'	'	'	'	'
TWINBIKE:			'	1	1	'												
SEATS:-1	-1	'-2		1	1	'			'									
WHEELS:-1	-2	'-2'			'1	1	'		'		'		'					
CHAINS:	-1	'-2					'1	1	'									
HUBS:							'-1	'		'1	1	'						
SPOKES:	'	'	'	'			'-B	'		'	'1	1	'					
LINKS:								'-B					'	'1	1	'		

The solution is:

Optimal solution found at step:	0	
Objective value:	3440.000	
Variable	Value	Reduced Cost
M(R)	500.0000	0.0000000
B(U)	100.0000	0.0000000
B(T)	200.0000	0.0000000
MM(S)	500.0000	0.0000000
MB(W)	1000.000	0.0000000
MB(C)	500.0000	0.0000000

Notice it is naturally integer. Thus, we should buy all the unicycles and twin bikes (and paste our own brand name on them). We assemble our own regular bicycles. They are assembled from manufactured seats and bought wheels and chains.

If we put an upper limit of 300 on the number of links manufactured by adding the constraint $LM \leq 300$, we will get a fractional solution because this constraint violates the MRP structure.

11.5.4 Transformations to Naturally Integer Formulations

A *row operation* consists of either of the following:

- multiplication through an equation by some non-zero constant,
- adding a finite multiple of one equation to another.

A row operation changes neither the feasible region nor the set of optimal solutions to a problem. Thus, if we can show a model can be transformed to either a network LP or an MRP LP by row operations, then we know there is an integer optimum. We do not actually need to do the transformation to get the solution.

Similarly, if we have a model with an integer right-hand side and we can show it is the dual of either a network LP or an MRP LP, then we know the model has an integer optimum.

Example

Consider the following LP that arose in planning how much to produce in each of four periods:

$$\begin{array}{cccccccccc}
 P & P & P & P & P & P & P & P & P & P \\
 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 4 \\
 4 & 3 & 2 & 1 & 4 & 3 & 2 & 4 & 3 & 4 \\
 1: & 9 & 6 & 4 & 3 & 6 & 4 & 3 & 4 & 3 & \text{MIN} \\
 2: & 1 & 1 & 1 & 1 & & & & & & = 1 \\
 3: & 1 & 1 & 1 & & 1 & 1 & 1 & & & = 1 \\
 4: & 1 & 1 & & 1 & 1 & & 1 & 1 & & = 1 \\
 5: & 1 & & 1 & & 1 & & 1 & & 1 & = 1
 \end{array}$$

When solved as an LP, we obtained the following naturally integer solution:

$$P_{12} = P_{34} = 1; \text{ all others } 0.$$

Could we have predicted a naturally integer solution beforehand? If we perform the row operations: $(5') = (5) - (4)$; $(4') = (4) - (3)$; $(3') = (3) - (2)$, we obtain the equivalent LP:

$$\begin{array}{cccccccccc}
 P & P & P & P & P & P & P & P & P & P \\
 1 & 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 4 \\
 4 & 3 & 2 & 1 & 4 & 3 & 2 & 4 & 3 & 4 \\
 1: & 9 & 6 & 4 & 3 & 6 & 4 & 3 & 4 & 3 & 3 \text{ MIN} \\
 2: & 1 & 1 & 1 & 1 & & & & & = 1 \\
 3': & & & -1 & 1 & 1 & 1 & & & = 0 \\
 4': & & -1 & & -1 & 1 & 1 & & & = 0 \\
 5': & -1 & & -1 & & -1 & 1 & & & = 0
 \end{array}$$

This is a network LP, so it has a naturally integer solution.

Example

In trying to find the minimum elapsed time for a certain project composed of seven activities, the following LP was constructed (in PICTURE form):

$$\begin{array}{lll}
 A & B & C & D & E & F \\
 1:-1 & ' & 1 & \text{MIN} \\
 AB:-1 & 1 & ' & & & \geq 3 \\
 AC:-1 & ' & 1 & ' & & \geq 2 \\
 BD: & -1 & 1 & & & \geq 5 \\
 BE: & -1 & ' & 1 & & \geq 6 \\
 CF: & ' & -1 & ' & 1 & \geq 4 \\
 DF: & & -1 & 1 & & \geq 7 \\
 EF: & & ' & -1 & 1 & \geq 6
 \end{array}$$

This is neither a network LP (e.g., consider columns A , B , or F) nor an MRP LP (e.g., consider columns A or F). Nevertheless, when solved, we get the naturally integer solution:

Optimal solution found at step:	0	
Objective value:	15.00000	
Variable	Value	
A	0.0000000	
B	3.0000000	
C	2.0000000	
D	8.0000000	
E	9.0000000	
F	15.00000	
Row	Slack or Surplus	
1	15.00000	
AB	0.0000000	
AC	0.0000000	
BD	0.0000000	
BE	0.0000000	
CF	9.0000000	
DF	0.0000000	
EF	0.0000000	
	Dual Price	
	1.0000000	
	-1.0000000	
	0.0000000	
	-1.0000000	
	0.0000000	
	-1.0000000	
	0.0000000	

Could we have predicted a naturally integer solution beforehand? If we look at the PICTURE of the model, we see each constraint has exactly one +1 and one -1. Thus, its dual model is a network LP and expectation of integer answers is justified.

11.6 The Assignment Problem and Related Sequencing and Routing Problems

The assignment problem is a simple LP problem, which is frequently encountered as a major component in more complicated practical problems.

The assignment problem is:

Given a matrix of costs:

c_{ij} = cost of assigning object i to person j ,

and variables:

$x_{ij} = 1$ if object i is assigned to person j .

Then, we want to:

Minimize $\sum_i \sum_j c_{ij}x_{ij}$

subject to

$\sum_i x_{ij} = 1$ for each object i ,

$\sum_j x_{ij} = 1$ for each person i ,

$x_{ij} \geq 0$.

This problem is easy to solve as an LP and the x_{ij} will be naturally integer.

There are a number of problems in routing and sequencing that are closely related to the assignment problem.

11.6.1 Example: The Assignment Problem

Large airlines tend to base their route structure around the hub concept. An airline will try to have a large number of flights arrive at the hub airport during a certain short interval of time (e.g., 9 A.M. to 10 A.M.) and then have a large number of flights depart the hub shortly thereafter (e.g., 10 A.M. to 11 A.M.). This allows customers of that airline to travel between a large combination of origin/destination cities with one stop and at most one change of planes. For example, United Airlines uses Chicago as a hub, Delta Airlines uses Atlanta, and American uses Dallas/Fort Worth.

A desirable goal in using a hub structure is to minimize the amount of changing of planes (and the resulting moving of baggage) at the hub. The following little example illustrates how the assignment model applies to this problem.

A certain airline has six flights arriving at O'Hare airport between 9:00 and 9:30 A.M. The same six airplanes depart on different flights between 9:40 and 10:20 A.M. The average numbers of people transferring between incoming and leaving flights appear below:

	L01	L02	L03	L04	L05	L06	
I01	20	15	16	5	4	7	
I02	17	15	33	12	8	6	
I03	9	12	18	16	30	13	
I04	12	8	11	27	19	14	Flight I05 arrives too late to
I05	0	7	10	21	10	32	connect with L01. Similarly I06 is
I06	0	0	0	6	11	13	too late for flights L01, L02, and L03.

All the planes are identical. A decision problem is which incoming flight should be assigned to which outgoing flight. For example, if incoming flight $I02$ is assigned to leaving flight $L03$, then 33 people (and their baggage) will be able to remain on their plane at the stop at O'Hare. How should incoming flights be assigned to leaving flights, so a minimum number of people need to change planes at the O'Hare stop?

This problem can be formulated as an assignment problem if we define:

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

The objective is to maximize the number of people not having to change planes. A formulation is:

```

MODEL: ! Assignment model (ASSIGNMX) ;
SETS:
  FLIGHT;
  ASSIGN( FLIGHT, FLIGHT): X, CHANGE;
ENDSETS
DATA:
  FLIGHT = 1..6;
! The value of assigning i to j;
  CHANGE = 20   15   16   5    4    7
            17   15   33   12   8    6
            9    12   18   16   30   13
           12    8   11   27   19   14
          -999   7   10   21   10   32
          -999 -999 -999   6   11   13;
ENDDATA
!-----;
! Maximize value of assignments;
MAX = @SUM(ASSIGN: X * CHANGE);
@FOR( FLIGHT( I):
  ! Each I must be assigned to some J;
  @SUM( FLIGHT( J): X( I, J)) = 1;
  ! Each I must receive an assignment;
  @SUM( FLIGHT( J): X( J, I)) = 1;
);
END

```

Notice, we have made the connections that are impossible prohibitively unattractive. A solution is:

```

Optimal solution found at step:      9
Objective value:                  135.0000
Variable          Value       Reduced Cost
X( 1, 1)        1.000000     0.0000000
X( 2, 3)        1.000000     0.0000000
X( 3, 2)        1.000000     0.0000000
X( 4, 4)        1.000000     0.0000000
X( 5, 6)        1.000000     0.0000000
X( 6, 5)        1.000000     0.0000000

```

Notice, each incoming flight except $I03$ is able to be assigned to its most attractive outgoing flight. The solution is naturally integer even though we did not declare any of the variables to be integer.

11.6.2 The Traveling Salesperson Problem

One of the more famous optimization problems is the traveling salesperson problem (TSP). It is an assignment problem with the additional condition that the assignments chosen must constitute a tour. The objective is to minimize the total distance traveled. Lawler et al. (1985) presents a tour-de-force on this fascinating problem. One example of a TSP occurs in the manufacture of electronic circuit boards. Danusaputro, Lee, and Martin-Vega (1990) discuss the problem of how to optimally sequence the drilling of holes in a circuit board, so the total time spent moving the drill head between holes is minimized. A similar TSP occurs in circuit board manufacturing in determining the sequence in which components should be inserted onto the board by an automatic insertion machine. Another example is the sequencing of cars on a production line for painting: each time there is a change in color, a setup cost and time is incurred.

A TSP is described by the data:

c_{ij} = cost of traveling directly from city i to city j , e.g., the distance.

A solution is described by the variables:

$y_{ij} = 1$ if we travel directly from i to j , else 0.

The objective is:

$$\text{Min } \sum_i \sum_j c_{ij} y_{ij};$$

We will describe several different ways of specifying the constraints.

Subtour Elimination Formulation:

- (1) We must enter each city j exactly once:

$$\sum_{i \neq j}^n y_{ij} = 1 \quad \text{for } j = 1 \text{ to } n,$$

- (2) We must exit each city i exactly once:

$$\sum_{j \neq i}^n y_{ij} = 1 \quad \text{for } i = 1 \text{ to } n,$$

- (3) $y_{ij} = 0$ or 1 , for $i = 1, 2, \dots, n$, $j = 1, 2, \dots, n$, $i \neq j$:

- (4) No subtours are allowed for any subset of cities S not including city 1:

$$\sum_{i,j \in S} y_{ij} \leq |S| - 1 \quad \text{for every subset } S,$$

where $|S|$ is the size of S .

The above formulation is usually attributed to Dantzig, Fulkerson, and Johnson(1954). An unattractive feature of the Subtour Elimination formulation is that if there are n cities, then there are approximately 2^n constraints.

Cumulative Load Formulation:

We can reduce the number of constraints substantially if we define: u_j = the sequence number of city j on the trip. Equivalently, if each city requires one unit of something to be picked up(or delivered), then u_j = cumulative number of units picked up(or delivered) after the stop at j . We replace constraint set (4) by:

$$(5) \quad u_j \geq u_i + 1 - (1 - y_{ij})n \quad \text{for } i = 1, 2, \dots, j = 2, 3, 4, \dots; j \neq i.$$

The approach of constraint set (5) is due to Miller, Tucker, and Zemlin(1960). There are only approximately n^2 constraints of type (5), however, constraint set (4) is much tighter than (5). Large problems may be computationally intractable if (4) is not used. Even though there are a huge number of constraints in (4), only a few of them may be binding at the optimum. Thus, an iterative approach that adds violated constraints of type (4) as needed works surprisingly well. Padberg and Rinaldi (1987) used essentially this iterative approach and were able to solve to optimality problems with over 2000 cities. The solution time was several hours on a large computer.

Multi-commodity Flow Formulation:

Similar to the previous formulation, imagine that each city needs one unit of some commodity distinct to that city. Define:

$$x_{ijk} = \text{units of commodity carried from } i \text{ to } j, \text{ destined for ultimate delivery to } k.$$

If we assume that we start at city 1 and there are n cities, then we replace constraint set (4) by:

For $k = 1, 2, 3, \dots, n$:

$$\sum_{j>1} x_{1jk} = 1; \quad (\text{Each unit must be shipped out of the origin.})$$

$$\sum_{i \neq k} x_{ikk} = 1; \quad (\text{Each city } k \text{ must get its unit.})$$

For $j = 2, 3, \dots, n, k = 1, 2, 3, \dots, n, j \neq k$:

$$\sum_i x_{ijk} = \sum_{t \neq j} x_{jtk} \quad (\text{Units entering } j, \text{ but not destined for } j, \text{ must depart } j \text{ to some city } t.)$$

A unit cannot return to 1, except if its final destination is 1:

$$\sum_i \sum_{k>1} x_{i1k} = 0,$$

For $i = 1, 2, \dots, n, j = 1, 2, \dots, n, k = 1, 2, \dots, n, i \neq j$:

$$x_{ijk} \leq y_{ij} \quad (\text{If anything shipped from } i \text{ to } j, \text{ then turn on } y_{ij}.)$$

The drawback of this formulation is that it has approximately n^3 constraints and variables. A remarkable feature of the multicommodity flow formulation is that it is just as tight as the Subtour Elimination formulation. The multi-commodity formulation is due to Claus(1984).

Heuristics

For practical problems, it may be important to get good, but not necessarily optimal, answers in just a few seconds or minutes rather than hours. The most commonly used heuristic for the TSP is due to Lin and Kernighan (1973). This heuristic tries to improve a given solution by clever re-orderings of cities in the tour. For practical problems (e.g., in operation sequencing on computer controlled machines), the heuristic seems always to find solutions no more than 2% more costly than the optimum. Bland and Shaller (1989) describe problems with up to 14,464 “cities” arising from the sequencing of operations on a computer controlled machine. In no case was the Lin-Kernighan heuristic more than 1.7% from the optimal for these problems.

Example of a Traveling Salesperson Problem

P. Rose, currently unemployed, has hit upon the following scheme for making some money. He will guide a group of 18 people on a tour of all the baseball parks in the National League. He is betting his life savings on this scheme, so he wants to keep the cost of the tour as low as possible. The tour will start and end in Cincinnati. The following distance matrix has been constructed:

	Atl	Chi	Cin	Hou	Lax	Mon	NYk	Phi	Pit	StL	SnD	SnF
Atlanta	0	702	454	842	2396	1196	864	772	714	554	2363	2679
Chicago	702	0	324	1093	2136	764	845	764	459	294	2184	2187
Cinci.	454	324	0	1137	2180	798	664	572	284	338	2228	2463
Houston	842	1093	1137	0	1616	1857	1706	1614	1421	799	1521	2021
L.A.	2396	2136	2180	1616	0	2900	2844	2752	2464	1842	95	405
Montreal	1196	764	798	1857	2900	0	396	424	514	1058	2948	2951
New York	864	845	664	1706	2844	396	0	92	386	1002	2892	3032
Phildpha.	772	764	572	1614	2752	424	92	0	305	910	2800	2951
Pittsbrg.	714	459	284	1421	2464	514	386	305	0	622	2512	2646
St. Louis	554	294	338	799	1842	1058	1002	910	622	0	1890	2125
San Diego	2363	2184	2228	1521	95	2948	2892	2800	2512	1890	0	500
San Fran.	2679	2187	2463	2021	405	2951	3032	2951	2646	2125	500	0

Solution

We will illustrate the subtour elimination approach, exploiting the fact that the distance matrix is symmetric. Define the decision variables:

$Y_{ij} = 1$ if the link between cities i and j is used, regardless of the direction of travel; 0 otherwise.

Thus, $Y(CHI, ATL) = 1$ if the link between Chicago and Atlanta is used. Each city or node must be connected to two links. In words, the formulation is:

Minimize the cost of links selected

subject to:

For each city, the number of links connected to it that are selected = 2

Each link can be selected at most once.

The LINGO formulation is shown below:

```

MODEL:
SETS:
  CITY;
  ROUTE(CITY, CITY) |&1 #GT# &2:COST, Y;
ENDSETS
DATA:
  CITY=
    ATL  CHI  CIN  HOU    LA  MON    NY  PHI  PIT  STL  SD  SF;
  COST=
    702
    454  324
    842  1093 1137
    2396 2136 2180 1617
    1196 764  798 1857 2900
    864  845  664 1706 2844  396
    772  764  572 1614 2752  424   92
    714  459  284 1421 2464  514  386  305
    554  294  338 799 1842 1058 1002  910  622
    2363 2184 2228 1521    95 2948 2892 2800 2512 1890
    2679 2187 2463 2021   405 2951 3032 2951 2646 2125 500;
ENDDATA
  MIN = @SUM(ROUTE: Y * COST);
  @SUM( CITY(I)|I #GE# 2: Y(I, 1)) = 2;
  @FOR( CITY(J)|J #GE# 2: @SUM(CITY(I)| I #GT# J:
    Y(I, J)) + @SUM(CITY(K)|K #LT# J: Y(J, K))=2);
  @FOR( ROUTE: Y <= 1);
END

```

When this model is solved *as an LP*, we get the solution:

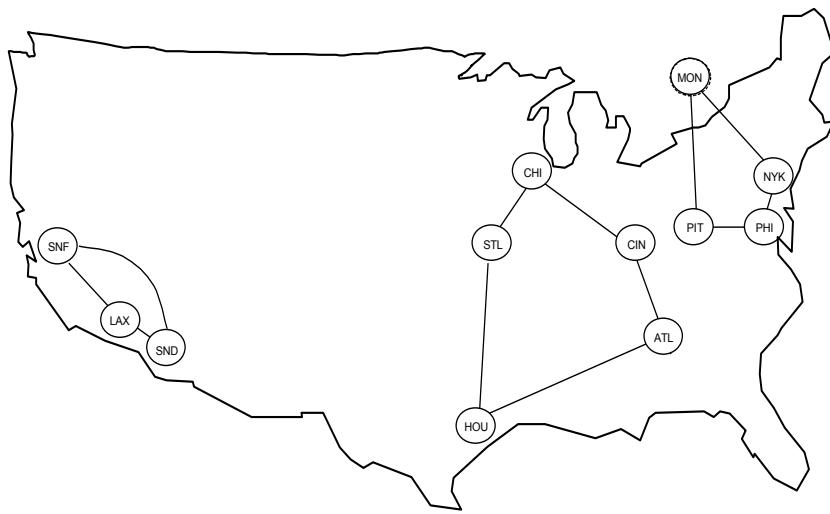
```

Optimal solution found at step:      105
Objective value:                  5020.000
Variable          Value
Y( CIN, ATL)      1.000000
Y( CIN, CHI)      1.000000
Y( HOU, ATL)      1.000000
Y( NYK, MON)      1.000000
Y( PHI, NYK)      1.000000
Y( PIT, MON)      1.000000
Y( PIT, PHI)      1.000000
Y( STL, CHI)      1.000000
Y( STL, HOU)      1.000000
Y( SND, LAX)      1.000000
Y( SNF, LAX)      1.000000
Y( SNF, SND)      1.000000

```

This has a cost of 5020 miles. Graphically, it corresponds to Figure 11.7.

Figure 11.7



Unfortunately, the solution has three subtours. We would like to cut off the smallest subtour by adding the constraint that looks like:

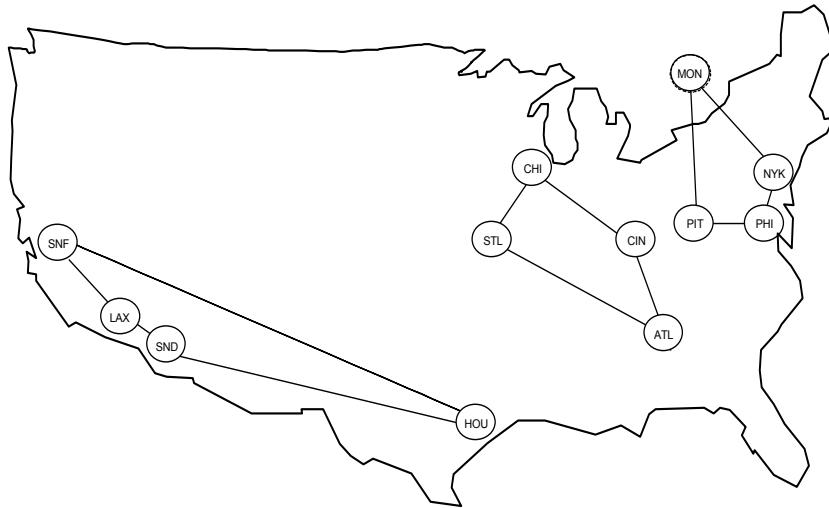
```
!SUBTOUR ELIMINATION;
Y( SNF, LAX) + Y( SND, LAX) + Y( SNF, SND) <= 2;
```

LINGO, however, works only with numeric subscripts, so if we want to use subscripts like SNF and LAX, we have to first tell LINGO their index values. The follow statements in the LINGO model equations will do this.

```
! A Trick: To make it easier to add problem specific cuts, give ourselves
some contants equal to index number of city with same name;
ATL=1; CHI=2; CIN=3; HOU=4; LAX=5; MON=6;
NYK=7; PHI=8; PIT=9; STL=10; SND=11; SNF=12;
! A longer, less clever approach uses the @INDEX() function.
E.g., LAX = @INDEX(LAX) would achieve the same effect.
Now we can add cuts, using names directly.;
```

Now, when we solve it *as an LP*, we get a solution with cost 6975, corresponding to Figure 11.8:

Figure 11.8



We cut off the subtour in the southwest by appending the constraint that says at most 3 arcs can be used involving the cities HOU, LAX, SND, and SNF:

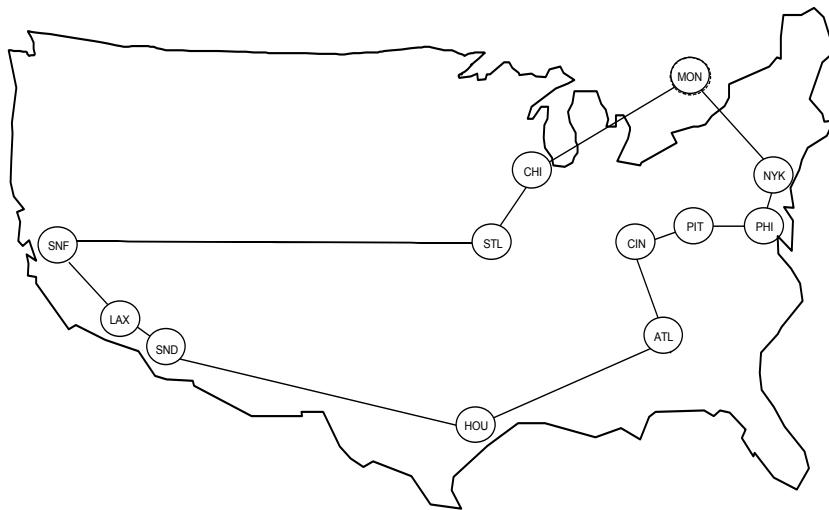
$$\begin{aligned} & Y(LAX, HOU) + Y(SND, HOU) + Y(SNF, HOU) \\ & + Y(SND, LAX) + Y(SNF, LAX) + Y(SNF, SND) \leq 3; \end{aligned}$$

We continue in this fashion appending subtour elimination cuts:

$$\begin{aligned} & Y(NYK, MON) + Y(PHI, MON) + Y(PIT, MON) + \\ & Y(PHI, NYK) + Y(PIT, NYK) + Y(PIT, PHI) \leq 3; \\ & Y(NYK, MON) + Y(PHI, MON) + Y(PHI, NYK) \leq 2; \end{aligned}$$

After the above are all appended, we get the solution shown in Figure 11.9. It is a complete tour with cost \$7,577.

Figure 11.9



Note only LPs were solved. No branch-and-bound was required, although in general branching may be required.

Could P. Rose have done as well by trial and error? The most obvious heuristic is the “closest unvisited city” heuristic. If one starts in Cincinnati and next goes to the closest unvisited city at each step and finally returns to Cincinnati, the total distance is 8015 miles, about 6% worse than the optimum.

The Optional Stop TSP

If we drop the requirement that every stop must be visited, we then get the optional stop TSP. This might correspond to a job sequencing problem where v_j is the profit from job j if we do it and c_{ij} is the cost of switching from job i to job j . Let:

$$y_j = 1 \text{ if city } j \text{ is visited, } 0 \text{ otherwise.}$$

If v_j is the value of visiting city j , then the objective is:

$$\text{Minimize } \sum_i \sum_j c_{ij} x_{ij} - \sum v_j y_j .$$

The constraint sets are:

- (1) Each city j can be visited at most once

$$\sum_{i \neq j} x_{ij} = y_j$$

- (2) If we enter city j , then we must exit it:

$$\sum_{k \neq j} x_{jk} = y_j$$

- (3) No subtours allowed for each subset, S , of cities not including the home base 1.

$$\sum_{i, j \in S} x_{ij} \leq |S| - 1, \text{ where } |S| \text{ is the size of } S.$$

For example, if there are n cities, including the home base, then there are $(n-1)(n-2)/(3 \times 2)$ subsets of size 3.

- (4) Alternatively, (3) may be replaced by

$$u_j \geq u_i + 1 - (1 - x_{ij})n \quad \text{for } j = 2, 3, \dots, n.$$

Effectively, u_j is the sequence number of city j in its tour. Constraint set (3) is much tighter than (4).

11.6.3 Capacitated Multiple TSP/Vehicle Routing Problems

An important practical problem is the routing of vehicles from a central depot. An example is the routing of delivery trucks for a metropolitan newspaper. You can think of this as a multiple traveling salesperson problem with finite capacity for each salesperson. This problem is sometimes called the LTL(Less than TruckLoad) routing problem because a typical recipient receives less than a truck load of goods. A formulation is:

Given:

V = capacity of a vehicle

d_j = demand of city or stop j

Each city, j , must be visited once for $j > 1$:

$$\sum_i x_{ij} = 1$$

Each city $i > 1$, must be exited once:

$$\sum_i x_{ij} = 1$$

No subtours:

$$\sum_{i, j \in S} x_{ij} \leq |S| - 1,$$

No overloads: For each set of cities T , including 1, which constitute more than a truckload:

$$\sum_{i, j \in T} x_{ij} \leq |T| - k,$$

where k = minimum number of cities that must be dropped from T to reduce it to one load.

This formulation can solve to optimality modest-sized problems of say, 25 cities. For larger or more complicated practical problems, the heuristic method of Clarke and Wright (1964) is a standard starting point for quickly finding good, but not necessarily optimal, solutions.

The following is a generic LINGO model for vehicle routing problems:

```

MODEL: ! (VROUTE);
! The Vehicle Routing Problem (VRP) occurs in many service
systems such as delivery, customer pick-up, repair and
maintenance. A fleet of vehicles, each with fixed
capacity, starts at a common depot and returns to the
depot after visiting locations where service is demanded.
Problems with more than a dozen cities can take lots of
time.

This instance involves delivering the required amount of
goods to 9 cities from a depot at city 1;

SETS:
CITY/ Chi Den Frsn Hous KC LA Oakl Anah Peor Phnx/: Q, U;
! Q(I) = amount required at city I(given),
must be delivered by just 1 vehicle.
U(I) = accumulated deliveries at city I ;
CXC( CITY, CITY): DIST, X;
! DIST(I,J) = distance from city I to city J
X(I,J) is 0-1 variable,
= 1 if some vehicle travels from city I to J,
else 0 ;
ENDSETS

DATA:
! city 1 represents the common depot, i.e. Q( 1) = 0;
Q= 0      6      3      7      7     18      4      5      2      6;
! distance from city I to city J is same from J to I,
distance from city I to the depot is 0,
because vehicle need not return to the depot ;
DIST= ! To City;
!Chi   Den   Frsn   Hous   KC   LA   Oakl   Anah   Peor   Phnx   From;
0    996  2162  1067  499  2054  2134  2050  151  1713! Chicago;
0     0  1167  1019  596  1059  1227  1055  904  792! Denver;
0  1167     0  1747  1723  214   168   250  2070  598! Fresno;
0  1019  1747     0  710  1538  1904  1528  948  1149! Houston;
0  596  1723   710     0  1589  1827  1579  354  1214! K. City;
0  1059   214  1538  1589     0  371   36  1943  389! L. A.;
0  1227   168  1904  1827  371     0  407  2043  755! Oakland;
0  1055   250  1528  1579   36   407     0  1933  379! Anaheim;
0  904  2070   948   354  1943  2043  1933     0  1568! Peoria;
0  792   598  1149  1214   389   755   379  1568     0;! Phoenix;

! VCAP is the capacity of a vehicle ;
VCAP = 18;
ENDDATA
!-----
! The objective is to minimize total travel distance;
MIN = @SUM( CXC: DIST * X);
! for each city, except depot....;
@FOR( CITY( K) | K #GT# 1:
    ! a vehicle does not travel inside itself....;
    X( K, K) = 0;

```

```

! a vehicle must enter it, ... ;
@SUM( CITY( I) | I #NE# K #AND# ( I #EQ# 1 #OR#
    Q( I) + Q( K) #LE# VCAP): X( I, K)) = 1;
    ! a vehicle must leave it after service ;
@SUM( CITY( J) | J #NE# K #AND# ( J #EQ# 1 #OR#
Q( J) + Q( K) #LE# VCAP): X( K, J)) = 1;
    ! U( K) = amount delivered on trip up to city K
>= amount needed at K but <= vehicle capacity;
@BND( Q( K), U( K), VCAP);

! If K follows I, then can bound U( K) - U( I);
@FOR( CITY( I) | I #NE# K #AND# I #NE# 1: U( K) >=
    U( I) + Q( K) - VCAP + VCAP*( X( K, I) + X( I, K))
    - ( Q( K) + Q( I)) * X( K, I);
);

! If K is 1st stop, then U( K) = Q( K);
U( K) <= VCAP - ( VCAP - Q( K)) * X( 1, K);
! If K is not 1st stop...;
U( K) >=
Q( K)+ @SUM( CITY( I) | I #GT# 1: Q( I) * X( I, K));
);

! Make the X's binary;
@FOR( CXC( I, J): @BIN( X( I, J)) );
;

! Must send enough vehicles out of depot;
@SUM( CITY( J) | J #GT# 1: X( 1, J)) >=
@FLOOR((@SUM( CITY( I) | I #GT# 1: Q( I))/ VCAP) + .999);
END

```

Optimal solution found at step: 973
 Objective value: 6732.000

Variable	Value
X(CHI, HOUS)	1.000000
X(CHI, LA)	1.000000
X(CHI, PEOR)	1.000000
X(CHI, PHNX)	1.000000
X(DEN, CHI)	1.000000
X(FRSN, OAKL)	1.000000
X(HOUS, CHI)	1.000000
X(KC, DEN)	1.000000
X(LA, CHI)	1.000000
X(OAKL, CHI)	1.000000
X(ANAH, FRSN)	1.000000
X(PEOR, KC)	1.000000
X(PHNX, ANAH)	1.000000

By following the links, you can observe that the trips are:

Chicago - Houston;
 Chicago - LA;
 Chicago - Peoria - KC - Denver;
 Chicago - Phoenix - Anaheim - Fresno - Oakland.

Combined DC Location/Vehicle Routing

Frequently, there is a vehicle routing problem associated with opening a new plant or distribution center (DC). Specifically, given the customers to be served from the DC, what trips are made, so as to serve the customers at minimum cost. A “complete” solution to the problem would solve the location and routing problems simultaneously. The following IP formulation illustrates one approach:

Parameters

F_i = fixed cost of having a DC at location i ,

C_j = cost of using route j ,

a_{ijk} = 1 if route j originates at DC i and serves customer k . There is exactly one DC associated with each route.

Decision variables

$y_i = 1$ if we use DC i , else 0,

$x_j = 1$ if we use route j , else 0

The Model

$$\text{Minimize } \sum_i F_i y_i + \sum_j c_j x_j$$

subject to

(Demand constraints)

For each customer k :

$$\sum_i \sum_j a_{ijk} x_j = 1$$

(Forcing constraints)

For each DC i and customer k :

$$\sum_j a_{ijk} x_j \leq y_i$$

11.6.4 Minimum Spanning Tree

A spanning tree of n nodes is a collection of $n - 1$ arcs, so there is exactly one path between every pair of nodes. A minimum cost spanning tree might be of interest, for example, in designing a communications network.

Assume node 1 is the root of the tree. Let $x_{ij} = 1$ if the path from 1 to j goes through node i immediately before node j , else $x_{ij} = 0$.

A formulation is:

$$\text{Minimize } \sum_i \sum_j c_{ij} x_{ij}$$

subject to

$$(1) \sum_i \sum_j x_{ij} = n - 1,$$

$$(2) \sum_{i, j \in S} x_{ij} \leq |S| - 1 \text{ for every strict subset } S \text{ of } \{1, 2, \dots, n\},$$

$$x_{ij} = 0 \text{ or } 1.$$

An alternative to (1) and (2) is the following set of constraints based on assigning a unique sequence number u_j to each node:

$$\begin{aligned} \sum_{i \neq j} x_{ij} &= 1, \text{ for } j = 2, 3, 4, \dots, n, \\ u_j &\geq u_i + x_{ij} - (n-2)(1-x_{ij}) + (n-3)x_{ji}, \text{ for } j = 2, 3, 4, \dots, n. \\ u_j &\geq 0. \end{aligned}$$

In this case, u_j is the number of arcs between node j and node 1. A numeric example of the sequence numbering formulation is in section 8.9.8.

If one has a pure spanning tree problem, then the “greedy” algorithm of Kruskal (1956) is a fast way of finding optimal solutions.

11.6.5 The Linear Ordering Problem

A problem superficially similar to the TSP is the linear ordering problem. One wants to find a strict ordering of n objects. Applications are to ranking in sports tournaments, product preference ordering in marketing, job sequencing on one machine, ordering of industries in an input-output matrix, ordering of historical objects in archeology, and others. See Grötschel et al. (1985) for a further discussion. The linear ordering problem is similar to the approach of conjoint analysis sometimes used in marketing. The crucial input data are cost entries c_{ij} . If object i appears *anywhere* before object j in the proposed ordering, then c_{ij} is the resulting cost. The decision variables are:

$$x_{ij} = 1 \text{ if object } i \text{ precedes object } j, \text{ either directly or indirectly for all } i \neq j.$$

The problem is:

$$\text{Minimize } \sum_i \sum_j c_{ij} x_{ij}$$

subject to

$$(1) x_{ij} + x_{ji} = 1 \text{ for all } i \neq j$$

If i precedes j and j precedes k , then we want to imply that i precedes k . This is enforced with the constraints:

$$(2) x_{ij} + x_{jk} + x_{ki} \leq 2 \text{ for all } i, j, k \text{ with } i \neq j, i \neq k, j \neq k.$$

The size of the formulation can be cut in two by noting that $x_{ji} = 1 - x_{ij}$. Thus, we substitute out x_{ji} for $j > i$. Constraint set (1) becomes simply $0 \leq x_{ij} \leq 1$. Constraint set (2) becomes:

$$\begin{aligned} (2') x_{ij} + x_{jk} - x_{ik} + s_{ijk} &= 1 \text{ for all } i < j < k \\ 0 \leq s_{ijk} &\leq 1 \end{aligned}$$

There are $n!/(n-3)! 3! = n \times (n-1) \times (n-2)/6$ ways of choosing 3 objects from n , so the number of constraints is approximately $n^3/6$.

Example

Ten Czech, German, and North American beverages were subject to taste tests by unbiased German testers. Each of the $10 \times 9/2 = 45$ possible pairs was subject to a taste test by six judges. The element $C(I, J)$ in the C matrix in the model below is the number of times out of six beverage I was preferred to beverage J . If we want to have a complete ranking for the beverages, a reasonable objective is to maximize the number of pairwise comparisons for which our ranking agrees with the pairwise ranking of the judges:

```

MODEL:
! Linear ordering of objects or products,
    based on pairwise comparisons(LINERORD);
SETS:
PROD: RANK; ! Each product will get a rank;
PXP( PROD, PROD): C;
ENDSETS
DATA:
PROD = KONIG, FURST, PILSURQ, GUNZB, RIEGELE,
       PAULA, JEVER, BECKS, WARST, BUD;
! Some data on German beverages;
C= ! Times that object I was preferred over J;
0   2   2   3   3   5   5   5   4   4
4   0   3   3   4   3   2   3   2   2
4   3   0   3   5   4   3   2   4   4
3   3   3   0   5   6   3   4   4   3
3   2   1   1   0   1   4   4   5   3
1   3   2   0   5   0   5   4   1   4
1   4   3   3   2   1   0   2   1   3
1   3   4   2   2   2   4   0   4   2
2   4   2   2   1   5   5   2   0   4
2   4   2   3   3   2   3   4   2   0;
ENDDATA
!-----
SETS:
PIP( PROD, PROD) | &1 #LT# &2:
    X; ! X(I,J) = 1 if I precedes J in our ranking;
PIPIP( PROD, PROD, PROD)
    | &1 #LT# &2 #AND# &2 #LT# &3: S;
ENDSETS
! Maximize the number of times our pairwise
    ordering matches that of our testers;
MAX =
@SUM( PIP( I, J): C( I, J) * X( I, J)
      + C( J, I) * (1 - X( I, J)));
! The rankings must be transitive, that is,
    If I->J and J->K, then I->K;
@FOR( PIP( I, J, K):
    ! Note N*(N-1)*(N-2)/6 of these!;
    X( I, J) + X( J, K) - X( I, K)
        + S( I, J, K) = 1;
    @BND( 0, S( I, J, K), 1);
);
@FOR( PIP: @BIN( X)); ! Make X's 0 or 1;

```

```

! Count number products before product I( + 1);
@FOR( PROD( I):
      RANK( I) = 1 + @SUM( PIP( K, I): X( K, I))
                  + @SUM( PIP( I, K): 1 - X( I, K));
   );
END

```

When solved, we get an optimal objective value of 168. This means out of the $(10 * 9/2)* 6 = 270$ pairwise comparisons, the pairwise rankings agreed with LINGO's complete ranking 168 times:

Optimal solution found at step:	50	
Objective value:	168.0000	
Branch count:	0	
Variable	Value	Reduced Cost
RANK(KONIG)	3.000000	0.0000000
RANK(FURST)	10.000000	0.0000000
RANK(PILSURQ)	2.000000	0.0000000
RANK(GUNZB)	1.000000	0.0000000
RANK(RIEGELE)	7.000000	0.0000000
RANK(PAULA)	5.000000	0.0000000
RANK(JEVER)	9.000000	0.0000000
RANK(BECKS)	8.000000	0.0000000
RANK(WARST)	4.000000	0.0000000
RANK(BUD)	6.000000	0.0000000

According to this ranking, *GUNZB* comes out number 1 (most preferred), while *FURST* comes out tenth (least preferred). It is important to note that there may be alternate optima. This means there may be alternate orderings, all of which match the input pairings 168 times out of 270. In fact, you can show that there is another ordering with a value of 168 in which *PILSURQ* is ranked first.

11.6.6 Quadratic Assignment Problem

The quadratic assignment problem has the same constraint set as the linear assignment problem. However, the objective function contains products of two variables. Notationally, it is:

$$\text{Min } \sum_{i} \sum_{j} \sum_{k} \sum_{l} c_{ijkl} x_{ij} x_{kl}$$

subject to:

For each j :

$$\sum_i x_{ij} = 1$$

For each i :

$$\sum_j x_{ij} = 1$$

Some examples of this problem are:

- (a) *Facility layout.* If d_{jl} is the physical distance between room j and room l ; s_{ik} is the communication traffic between department i and k ; and $x_{ij} = 1$ if department i is assigned to room j , then we want to minimize:

$$\sum_{i} \sum_{j} \sum_{k} \sum_{l} x_{ij} x_{kl} d_{jl} s_{ik}$$

- (b) *Vehicle to gate assignment at a terminal.* If d_{jl} is the distance between gate j and gate l at an airline terminal, passenger train station, or at a truck terminal; s_{ik} is the number of passengers or tons of cargo that needs to be transferred between vehicle i and vehicle k ; and $x_{ij} = 1$ if vehicle i (incoming or outgoing) is assigned to gate j , then we again want to minimize:

$$\sum_{i} \sum_{j} \sum_{k} \sum_{l} x_{ij} x_{kl} d_{jl} s_{ik}$$

- (c) *Radio frequency assignment.* If d_{ij} is the physical distance between transmitters i and j ; s_{kl} is the distance in frequency between k and l ; and p_i is the power of transmitter i , then we want $c_{ijkl} = \max\{p_i, p_j\} (1/d_{ij})(1/s_{kl})$ to be small if transmitter i is assigned frequency k and transmitter j is assigned frequency l .

- (d) *VLSI chip layout.* The initial step in the design of a VLSI (very large scale integrated) chip is typically to assign various required components to various areas on the chip. See Sarrafzadeh and Wong (1996) for additional details. Steinberg (1961) describes the case of assigning electronic components to a circuit board, so as to minimize the total interconnection wire length. For the chip design case, typically the chip area is partitioned into 2 to 6 areas. If d_{jl} is the physical distance between area j and area l ; s_{ik} is the number of connections required between components i and k ; and $x_{ij} = 1$ if component i is assigned to area j , then we again want to minimize:

$$\sum_{i} \sum_{j} \sum_{k} \sum_{l} x_{ij} x_{kl} d_{jl} s_{ik}$$

- (e) *Disk file allocation.* If w_{ij} is the interference if files i and j are assigned to the same disk, we want to assign files to disks, so total interference is minimized.

- (f) *Type wheel design.* Arrange letters and numbers on a type wheel, so (a) most frequently used ones appear together and (b) characters that tend to get typed together (e.g., q u) appear close together on the wheel.

The quadratic assignment problem is a notoriously difficult problem. If someone asks you to solve such a problem, you should make every effort to show the problem is not really a quadratic assignment problem. One indication of its difficulty is the solution is not naturally integer.

One of the first descriptions of quadratic assignment problems was by Koopmans and Beckmann (1957). For this reason, this problem is sometimes known as the Koopmans-Beckmann problem. They illustrated the use of this model to locate interacting facilities in a large country. Elshafei (1977) illustrates the use of this model to lay out a hospital. Specifically, 19 departments are assigned to 19 different physical regions in the hospital. The objective of Elshafei was to minimize the total distance patients had to walk between departments. The original assignment used in the hospital required a distance of 13,973,298 meters per year. An optimal assignment required a total distance of 8,606,274 meters. This is a reduction in patient travel of over 38%.

Small quadratic assignment problems can be converted to linear integer programs by the transformation:

Replace the product $x_{ij} x_{kl}$ by the single variable z_{ijkl} . The objective is then:

$$\text{Min } \sum_{i} \sum_{j} \sum_{k} \sum_{l} c_{ijkl} z_{ijkl}$$

Notice if there are N departments and N locations, then there are $N \times N$ variables of type x_{ij} , and $N \times N \times N \times N$ variables of type z_{ijkl} variables. This formulation can get large quickly. Several reductions are possible:

- 1) The terms $c_{ijkl} x_{ij} x_{kl}$ and $c_{klij} x_{kl} x_{ij}$ can be combined into the term:

$$(c_{ijkl} + c_{klij}) x_{kl} x_{ij}$$

to reduce the number of z variables and associated constraints needed by a factor of 2.

- 2) Certain assignments can be eliminated beforehand (e.g., a large facility to a small location). Many of the cross terms, c_{ijkl} , are zero (e.g., if there is no traffic between facility i and facility k), so the associated z variables need not be introduced.

The non-obvious thing to do now is to ensure that $z_{ijkl} = 1$ if and only if both $x_{ij} = 1$ and $x_{kl} = 1$. Sherali and Adams(1999) point out that constraints of the following type will enforce this requirement:

For a given i, k, l :

$$x_{kl} = \sum_{j, j \neq l} z_{ijkl}$$

In words, if object k is assigned to location l , then for any other object i , $i \neq k$, there must be some other location $j, j \neq l$, to which i is assigned.

The following is a LINGO implementation of the above for deciding which planes should be assigned to which gates at an airport, so that the distance weighted cost of changing planes for the passengers is minimized:

```

MODEL:
! Quadratic assignment problem(QAP006);
! Given number of transfers between flights,
! distance between gates,
! assign flights to gates to minimize total transfer cost;
SETS:
FLIGHT/1..6/;
GATE/ E3 E4 E5 F3 F4 F5/;! Gates at terminal 2 of O'Hare;
GXG( GATE, GATE) | &1 #LT# &2: T; ! Inter gate times(symmetric);
FXF( FLIGHT, FLIGHT) | &1 #LT# &2: N; ! Transfers between flights;
FXG( FLIGHT, GATE): X; ! Flight to gate assignment variable;
ENDSETS
DATA:
T = 70 40 60 90 90 ! Time between gates;
      50 100 80 110
          100 90 130
          60 40
            30;

```

```

N = 12   0 12   0   5
      30 35 20 13 ! No. units between flights;
      40 20 10
          0   6
          14;

ENDDATA
!-----
! Warning: may be very slow for no. objects > 7;
SETS: ! Warning: this set gets big fast!;
    TGTG( FLIGHT, GATE, FLIGHT, GATE) | &1 #LT# &3: Z;
ENDSETS

! Min the cost of transfers * distance;
MIN = @SUM( TGTG( B, J, C, K) | J #LT# K:
            Z( B, J, C, K) * N( B, C) * T( J, K))
      + @SUM( TGTG( B, J, C, K) | J #GT# K:
            Z( B, J, C, K) * N( B, C) * T( K, J));

! Each flight, B, must be assigned to a gate;
@FOR( FLIGHT( B):
    @SUM( GATE( J): X( B, J)) = 1;
);

! Each gate, J, must receive one flight;
@FOR( GATE( J):
    @SUM( FLIGHT( B): X( B, J)) = 1;
);

! Make the X's binary;
@FOR( FXG: @BIN( X);
);

! Force the Z() to take the correct value relative to the X();
@FOR( FXG( C, K):
    @FOR( GATE( J) | J #NE# K:
        X( C, K) = @SUM( TGTG( B, J, C, K) | B #NE# C : Z( B, J, C, K))
                  + @SUM( TGTG( C, K, B, J) | B #NE# C : Z( C, K, B, J));
    );
    @FOR( FLIGHT( B) | B #NE# C:
        X( C, K) = @SUM( TGTG( B, J, C, K) | J #NE# K : Z( B, J, C, K))
                  + @SUM( TGTG( C, K, B, J) | J #NE# K : Z( C, K, B, J));
    );
);

END

```

The solution is:

```

Global optimal solution found at step:      1258
Objective value:                          13490.00
Branch count:                            0

Variable          Value
X( 1, E4)        1.000000
X( 2, F4)        1.000000
X( 3, F3)        1.000000
X( 4, F5)        1.000000
X( 5, E3)        1.000000
X( 6, E5)        1.000000

```

Thus, flight 1 should be assigned to gate $E4$, flight 2 to gate $F4$, etc. The total passenger travel time in making the connections will be 13,490. Notice that this formulation was fairly tight. No branches were required to get an integer solution from the LP solution.

11.7 Problems of Grouping, Matching, Covering, Partitioning, and Packing

There is a class of problems that have the following essential structure:

- 1) There is a set of m objects, and
- 2) They are to be grouped into subsets, so some criterion is optimized.

Some example situations are:

Objects	Group	Criteria for a Group
(i) Dormitory inhabitants	Roommates	At most two to a room; no smokers with nonsmokers.
(ii) Deliveries to customers	Trip	Total weight assigned to trip is less-than-or-equal-to vehicle capacity. Customers in same trip are close together.
(iii) Sessions at a scientific meeting	Sessions scheduled for same time slot	No two sessions on same general topic. Enough rooms of sufficient size.
(iv) Exams to be scheduled	Exams scheduled for same time slot	No student has more than one exam in a given time slot.
(v) Sportsmen	Foursome (e.g., in golf or tennis doubles).	Members are of comparable ability, appropriate combination of sexes as in tennis mixed doubles.
(vi) States on map to be colored.	All states of a given color	States in same group/color cannot be adjacent.
(vii) Finished good widths needed in a paper plant roll.	Widths cut from a single raw paper roll.	Sum of finished good widths must not exceed raw material width.
(viii) Pairs of points	Connection layers	Connection paths in a layer should not intersect.

to connect on a circuit board underneath the circuit board Total lengths of paths are small.

If each object can belong to at most one group, it is called a *packing* problem. For example, in a delivery problem, as in (ii) above, it may be acceptable that a low priority customer not be included in any trip today if we are confident the customer could be almost as well served by a delivery tomorrow. If each object must belong to exactly one group, it is called a *partitioning* problem. For example, in circuit board routing as in (vii) above, if a certain pair of points must be connected, then that pair of points must be assigned to exactly one connection layer underneath the board. If each object must belong to at least one group, it is called a *covering* problem. A packing or partitioning problem with group sizes limited to two or less is called a *matching* problem. Specialized and fast algorithms exist for matching problems. A problem closely related to covering problems is the cutting stock problem. It arises in paper, printing, textile, and steel industries. In this problem, we want to determine cutting patterns to be used in cutting up large pieces of raw material into finished-good-size pieces.

Although grouping problems may be very easy to state, it may be very difficult to find a provably optimal solution if we take an inappropriate approach. There are two common approaches to formulating grouping problems: (1) assignment style, or (2) the partition method. The former is convenient for small problems, but it quickly becomes useless as the number of objects gets large.

11.7.1 Formulation as an Assignment Problem

The most obvious formulation for the general grouping problem is based around the following definition 0/1 decision variables:

$$X_{ij} = 1 \text{ if object } j \text{ is assigned to group } i, 0 \text{ otherwise.}$$

A drawback of this formulation is that it has a lot of symmetry. There are many alternate optimal solutions. All of which essentially are identical. For example, assigning golfers *A*, *B*, *C*, and *D* to group 1 and golfers *E*, *F*, *G*, and *H* to group 2 is essentially the same as assigning golfers *E*, *F*, *G*, and *H* to group 1 and golfers *A*, *B*, *C* and *D* to group 2. These alternate optima make the typical integer programming algorithm take much longer than necessary.

We can eliminate this symmetry and the associated alternate optima with no loss of optimality if we agree to the following restrictions: (a) object 1 can only be assigned to group 1; (b) object 2 can only be assigned to groups 1 or 2 and only to 1 if object 1 is also assigned to 1; (c) and in general, object *j* can be assigned to group *i* $< j$, only if object *i* is also assigned to group *i*. This implies in particular that:

$$\begin{aligned} X_{ii} &= 1, \text{ if and only if object } i \text{ is the lowest indexed object in its group, and} \\ X_{ij} &\text{ is defined only for } i \leq j. \end{aligned}$$

Now we will look at several examples of grouping problems and show how to solve them.

11.7.2 Matching Problems, Groups of Size Two

The roommate assignment problem is a simple example of a grouping problem where the group size is two. An example of this is a problem solved at many universities at the start of the school year before the first-year or freshman students arrive. The rooms in a freshman dormitory typically take exactly two students. How should new incoming students be paired up? One approach that has been used is that for every possible pair of students, a score is calculated which is a measure of how well the school thinks this particular pair of students would fare as roommates. Considerations that enter into a

score are things such as: a smoker should not be matched with a nonsmoker, a person who likes to study late at night should not be paired with a student who likes to get up early and study in the morning. Let us suppose we have computed the scores for all possible pairs of the six students: Joe, Bob, Chuck, Ed, Evan, and Sean. A scalar model for this problem might be:

```

! Maximize total score of pairs selected;
MAX= 9*X_JOE_BOB + 7*X_JOE_CHUCK + 4*X_JOE_ED
+ 6*X_JOE_EVAN + 3*X_JOE_SEAN + 2*X_BOB_CHUCK
+ 8*X_BOB_ED + X_BOB_EVAN + 7*X_BOB_SEAN
+ 3*X_CHUCK_ED + 4*X_CHUCK_EVAN + 9*X_CHUCK_SEAN
+ 5*X_ED_EVAN + 5*X_ED_SEAN + 6*X_EVAN_SEAN;

! Each student must be in exactly one pair;
[JOE] X_JOE_BOB + X_JOE_CHUCK + X_JOE_ED
+ X_JOE_EVAN + X_JOE_SEAN = 1;
[BOB] X_JOE_BOB + X_BOB_CHUCK + X_BOB_ED
+ X_BOB_EVAN+ X_BOB_SEAN = 1;
[CHUCK] X_JOE_CHUCK + X_BOB_CHUCK + X_CHUCK_ED
+ X_CHUCK_EVAN+ X_CHUCK_SEAN = 1;
[ED] X_JOE_ED + X_BOB_ED + X_CHUCK_ED + X_ED_EVAN
+ X_ED_SEAN = 1;
[EVAN] X_JOE_EVAN + X_BOB_EVAN + X_CHUCK_EVAN + X_ED_EVAN
+ X_EVAN_SEAN = 1;
[SEAN] X_JOE_SEAN + X_BOB_SEAN + X_CHUCK_SEAN
+ X_ED_SEAN + X_EVAN_SEAN = 1;

! Assignments must be binary, not fractional;
@BIN( X_JOE_BOB); @BIN( X_JOE_CHUCK); @BIN( X_JOE_ED);
@BIN( X_JOE_EVAN); @BIN( X_JOE_SEAN); @BIN( X_BOB_CHUCK);
@BIN( X_BOB_ED); @BIN( X_BOB_EVAN); @BIN( X_BOB_SEAN);
@BIN( X_CHUCK_ED); @BIN( X_CHUCK_EVAN); @BIN( X_CHUCK_SEAN);
@BIN( X_ED_EVAN); @BIN( X_ED_SEAN); @BIN( X_EVAN_SEAN);

```

Notice that there is a variable `X_JOE_BOB`, but not a variable `X_BOB_JOE`. This is because we do not care whose name is listed first on the door. We only care about which two are paired together. We say we are interested in unordered pairs.

A typical dormitory may have 60, or 600, rather than 6 students, so a general, set based formulation would be useful. The following formulation shows how to do this in LINGO. One thing we want to do in the model is to tell LINGO that we do not care about the order of persons in a pair. LINGO conveniently allows us to put conditions on which of all possible members (pairs in this case) of a set are to be used in a specific model. The key statement in the model is:

```
PXP( PERSON, PERSON) | &1 #LT# &2: VALUE, X;
```

The fragment, `PXP(PERSON, PERSON)`, by itself, tells LINGO that the set `PXP` should consist of all possible combinations, 6×6 for this example, of two persons. The conditional phrase, `| &1 #LT#`

`&2`, however, tells LINGO to restrict the combinations to those in which the index number, `&1`, of the first person in a pair should be strictly less than the index number, `&2`, of the second person.

```

MODEL: ! (roomates.lng);
SETS:
PERSON;
! Joe rooms with Bob means the same as
Bob rooms with Joe, so we need only the
upper triangle;
PXP( PERSON, PERSON) | &1 #LT# &2: VALUE, X;
ENDSETS
DATA:
PERSON = Joe Bob Chuck Ed Evan Sean;
Value =         9      7      4      6      3      ! Joe;
              2      8      1      7      ! Bob;
              3      4      9      5      ! Chuck;
              5      5      ! Ed;
              6 ; ! Evan;
ENDDATA

! Maximize the value of the matchings;
MAX = @SUM( PXP(I,J): Value(i,j)* X(I,J));

! Each person appears in exactly one match;
@FOR( PERSON( K):
      @SUM( PXP(K,J): X(K,J)) + @SUM( PXP(I,K): X(I,K)) = 1;
);

! No timesharing;
@FOR( PXP(I,J): @BIN(X(I,J)));
END

```

The constraint, $\sum_{J} X(K,J) + \sum_{I} X(I,K) = 1$ has two terms, the first where student K is the first person in the pair, the second summation is over the variables where student K is the second person in the pair. For example, in the scalar formulation, notice that ED is the first person in two of the pairs, and the second person of three of the pairs.

The following solution, with value 23, is found.

Variable	Value
<code>X(JOE, EVAN)</code>	1.000000
<code>X(BOB, ED)</code>	1.000000
<code>X(CHUCK, SEAN)</code>	1.000000

So Joe is to be paired with Evan, Bob with Ed, and Chuck with Sean. This model scales up well in that it can be easily solved for large numbers of objects, e.g., many hundreds.

For a different perspective on matching, see the later section on “stable matching”.

11.7.3 Groups with More Than Two Members

The following example illustrates a problem recently encountered by an electricity generating firm and its coal supplier. You are a coal supplier and you have a nonexclusive contract with a consumer owned and managed electric utility, Power to the People (PTTP). You supply PTTP by barge. Your contract with PTTP stipulates that the coal you deliver must have at least 13000 BTU's per ton, no more than 0.63% sulfur, no more than 6.5% ash, and no more than 7% moisture. Historically, PTTP would not accept a barge if it did not meet the above requirements.

You currently have the following barge loads available.

Barge	BTU/ton	Sulfur%	Ash%	Moisture%
1	13029	0.57	5.56	6.2
2	14201	0.88	6.76	5.1
3	10630	0.11	4.36	4.6
4	13200	0.71	6.66	7.6
5	13029	0.57	5.56	6.2
6	14201	0.88	6.76	5.1
7	13200	0.71	6.66	7.6
8	10630	0.11	4.36	4.6
9	14201	0.88	6.76	5.1
10	13029	0.57	5.56	6.2
11	13200	0.71	6.66	7.6
12	14201	0.88	6.76	5.1

This does not look good. Only barges 1, 5, and 10 satisfy PTTP's requirement. What can we do? Suppose that after reading the fine print of your PTTP contract carefully, you initiate some discussions with PTTP about how to interpret the above requirements. There might be some benefits if you could get PTTP to reinterpret the wording of the contract so that the above requirements apply to collections of up to three barges. That is, if the average quality taken over a set of N barges, N less than four, meets the above quality requirements, then that set of N barges is acceptable. You may specify how the sets of barges are assembled. Each barge can be in at most one set. All the barges in a set must be in the same shipment.

Looking at the original data, we see, even though there are twelve barges, there are only four distinct barge types represented by the original first four barges. In reality, you would expect this: each barge type corresponding to a specific mine with associated coal type.

Modeling the barge grouping problem as an assignment problem is relatively straightforward. The essential decision variable is defined as $X(I, J)$ = number of barges of type I assigned to group J . Note we have retained the convention of not distinguishing between barges of the same type. Knowing there are twelve barges, we can restrict ourselves to at most six groups without looking further at the data. The reasoning is: Suppose there are seven nonempty groups. Then, at least two of the groups must be singletons. If two singletons are feasible, then so is the group obtained by combining them. Thus, we can write the following LINGO model:

```

MODEL:
SETS:
  MINE: BAVAIL;
  GROUP;
  QUALITY: QTARG;
!  Composition of each type of MINE load;
  MXQ( MINE, QUALITY): QACT;
!assignment of which MINE to which group;
!no distinction between types;
  MXG( MINE, GROUP):X;
ENDSETS
DATA:
  MINE = 1..4;
    ! Barges available of each type(or mine);
  BAVAIL = 3 4 2 3;
  QUALITY = BTU, SULF, ASH, MOIST;
    ! Quality targets as upper limits;
  QTARG = - 13000 0.63 6.5 7;
    ! Actual qualities of each mine;
  QACT = -13029 0.57 5.56 6.2
        -14201 0.88 6.76 5.1
        -10630 0.11 4.36 4.6
        -13200 0.71 6.66 7.6;
! We need at most six groups;
  GROUP = 1..6;
  GRPSIZ = 3;
ENDDATA
! Maximize no. of barges assigned;
MAX = @SUM( MXG: X);
! Upper limit on group size;
@FOR( GROUP(J): @SUM( MINE( I): X(I, J))
  <= GRPSIZ););
! Assign no more of a type than are available;
@FOR( MINE(I): @SUM( GROUP( J): X( I, J))
  <= BAVAIL( I));
! The blending constraints for each group;
@FOR( GROUP(J):
  @FOR( QUALITY ( H):
    @SUM( MINE( I): X( I, J) * QACT( I, H)) <=
    @SUM( MINE( I): X( I, J) * QTARG( H));
  ));
! barges must be integers;
@FOR( MXG: @GIN( X));
END

```

The following solution shows that you can now sell ten barges, rather than three, to PTTP.

Objective value:	10.00000
Variable	Value
X(1, 1)	1.000000
X(2, 2)	2.000000
X(3, 2)	1.000000
X(1, 4)	2.000000
X(4, 4)	1.000000
X(2, 5)	2.000000
X(3, 5)	1.000000

For example, group 1 is simply one barge of type 1. Group 2 consists two barges of type 2 and one barge of type 3. The above formulation may not scale well. The actual application typically had about 60 barges in a candidate set. The above formulation may be slow to solve problems of that size. The next section discusses how the partitioning approach can be efficiently used for such problems.

Solving with a Partitioning Formulation

Modest-sized integer programs can nevertheless be very difficult to solve. There are a number of rules that are useful when facing such problems. Two useful rules for difficult integer programs are:

- 1) Do Not Distinguish the Indistinguishable;
- 2) Presolve subproblems.

The barge matching example can be solved almost “by hand” with the matching or grouping (as opposed to the assignment) approach. Applying the rule “Presolve subproblems,” we can enumerate all feasible combinations of three or less barges selected from the four types. Applying the “Don’t distinguish” rule again, we do not have to consider combinations such as (1,1) and (2,2,2), because such sets are feasible if and only if the singleton sets (e.g., (1) and (2)) are also feasible. Thus, disregarding quality, there are four singleton sets, six doubleton sets, four distinct triplets (e.g., (1,2,3)) and twelve paired triplets (e.g., (1,1,2)) for a total of 26 combinations. It is not hard to show, even manually, that the only feasible combinations are (1), (1,1,4), and (2,2,3). Thus, the matching-like IP we want to solve to maximize the number of barges sold is:

```

Max = S001 + 3 * S114 + 3 * S223;
      S001 + 2 * S114      <= 3 ;
      !(No. of type 1 barges);
      2 * S223 <= 4 ;
      !(No. of type 2 barges);
      S223 <= 2 ;
      !(No. of type 3 barges);
      S114      <= 3 ;
      !(No. of type 4 barges);
  
```

This is easily solved to give $S001 = 1$, $S114 = 1$, and $S223 = 2$, with an objective value of 10.

For the given data, we can ship at most ten barges. One such way of matching them, so each set satisfies the quality requirements is as follows:

Barges in set	Average Quality of the Set			
	BTU%	Sulfur%	Ash%	Moisture%
1	13029	0.57	5.56	6.2
4, 5, 10	13086	0.6167	5.927	6.667
2, 3, 6	13010	0.6233	5.96	4.933
8, 9, 12	13010	0.6233	5.96	4.933

This matches our LINGO derived solution.

11.7.4 Groups with a Variable Number of Members, Assignment Version

In many applications of the grouping idea, the group size may be variable. The following example from the financial industry illustrates. A financial services firm has financial objects (e.g., mortgages) it wants to “package” and sell. One of the features of a package is that it must contain a combination of objects whose values total at least one million dollars. For our purposes, we will assume this is the only qualification in synthesizing a package. We want to maximize the number of packages we form. We first give an assignment formulation. The key declaration in this formulation is:

OXO(OBJECT, OBJECT) | &1 #LE# &2: X;

This implies there will be a variable of the form $X(I, J)$ with always the index $I \leq J$. Our interpretation of this variable will be:

$X(I, J) = 1$ means object J is assigned to the same group as object I , and further,

$X(I, I) = 1$ means object I is the lowest indexed object in that group.

```

MODEL:
! Object bundling model. (OBJBUNDL);
!A broker has a number of loans of size from $55,000 to $946,000.
The broker would like to group the loans into packages
so that each package has at least $1M in it,
and the number of packages is maximized;
! Keywords: bundling, financial, set packing;
SETS:   OBJECT: VALUE, OVER;
        OXO( OBJECT, OBJECT) | &1 #LE# &2: X;
ENDSETS
DATA:
OBJECT= A   B   C   D   E   F   G   H   I   J   K   L   M   N   P   Q   R;
      VALUE=910 870 810 640 550 250 120 95 55 200 321 492 567 837 193 364 946;
! The value in each bundle must be >= PKSIZE;
      PKSIZE = 1000;
ENDDATA
!-----
! Definition of variables;
! X( I, I) = 1 if object I is lowest numbered
          object in its package;
! X( I, J) = 1 if object j is assigned to package I;
! Maximize number of packages assembled;
MAX = @SUM( OBJECT( I): X( I, I));

```

```

@FOR( OBJECT( K):
! Each object can be assigned to at most one package;
  @SUM( OXO( I, K): X( I, K)) <= 1;
! A package must be at least PSIZE in size;
  @SUM( OXO( K, J): VALUE( J) * X( K, J))
    - OVER( K) = PKSIZE * X( K, K);
  );
! The X( I, J) must = 0 or 1;
  @FOR( OXO( I, J): @BIN( X( I, J)););
END

```

A solution is:

Variable	Value
X(A, A)	1.000000
X(A, H)	1.000000
X(B, B)	1.000000
X(B, F)	1.000000
X(C, C)	1.000000
X(C, J)	1.000000
X(D, D)	1.000000
X(D, Q)	1.000000
X(E, E)	1.000000
X(E, L)	1.000000
X(G, G)	1.000000
X(G, K)	1.000000
X(G, M)	1.000000
X(I, I)	1.000000
X(I, R)	1.000000
X(N, N)	1.000000
X(N, P)	1.000000

Thus, eight packages are constructed. Namely: *AH*, *BF*, *CJ*, *DQ*, *EL*, *IR*, *JN*, *GKM*, and *NP*. It happens that every object appears in some package. There are alternate packings of all the objects into eight groups. Thus, one may wish to consider secondary criteria for choosing one alternate optimum over another (e.g., the largest package should be as close as possible to one million in size). The worst package in the fairness sense in the above solution is *BF*. It is over the target of 1,000,000 by 120,000.

11.7.5 Groups with A Variable Number of Members, Packing Version

An alternative approach is first to enumerate either all possible or all interesting feasible groups and then solve an optimization problem of the form:

Maximize value of the groups selected
 subject to:
 Each object is in at most one of the selected groups.

The advantage of this formulation is, when it can be used, it typically can be solved more easily than the assignment formulation. The disadvantages are it may have a huge number of decision variables, especially if the typical group size is more than three. If there are n distinct objects, and all groups are of size k , then there are $n!/(k!(n-k)!)$ distinct groups. For example, if $n = 50$ and $k = 3$, then there are 19,600 candidate groups.

This formulation uses the idea of composite variables. This is frequently a useful approach for a problem for which the original or “natural” formulation is difficult to solve. Setting a particular composite variable to 1 represents setting a particular combination of the original variables to 1. We generate only those composite variables that correspond to feasible combinations of the original variables. This effectively eliminates many of the fractional solutions that would appear if one solved the LP relaxation of the original formulation. The composite variable idea is a form of what is sometimes called column generation. The path formulation in network models is also an example of the use of composite variables.

Example: Packing Financial Instruments, revisited.

The packing approach to formulating a model for this problem constructs all possible packages or groups that just satisfy the one million minimum. The general form of the LP/IP is:

Maximize value of packages selected
 subject to:
 Each object appears in at most one selected package.

In the formulation below, we will use sparse sets to represent our packages. We assume that we need not consider packages of more than four objects. An attractive feature of the packing/partitioning formulation is that we can easily attach an essentially arbitrary score to each possible group. In particular, the following formulation applies a squared penalty to the extent to which a package of loans exceeds the target of \$1M.

```

MODEL:
! Object bundling model. (OBJBUNDH);
! A broker has a number of loans of size from $55,000 to
$946,000.
The broker would like to group the loans into packages
so that each package has at least $1M in it, preferably
not much more,
and the number of packages is maximized;
! Keywords: bundling, financial, set packing;
SETS:
OBJECT: VALUE;
ENDSETS
DATA:
OBJECT = A   B   C   D   E   F   G   H   I   J   K   L   M   N   P   Q   R;
VALUE = 910 870 810 640 550 250 120  95  55 200 321 492 567 837 193 364
946;
! The value in each bundle must be >= PKSIZE;
PKSIZE = 1000;
ENDDATA
SETS:
! Enumerate all 2,3, and 4 object unordered sets ≤ package
size;
BNDL2( OBJECT, OBJECT) | &1 #LT# &2
#AND# (VALUE(&1) + VALUE(&2)) #GE# PKSIZE: X2, OVER2;
BNDL3( OBJECT, OBJECT, OBJECT) | &1 #LT# &2 #AND# &2 #LT#
&3

```

```

#AND# ( VALUE(&1) + VALUE(&2) + VALUE(&3) #GE# PKSIZE):
    X3, OVER3;
BNDL4( OBJECT, OBJECT, OBJECT, OBJECT) | &1 #LT# &2
    #AND# &2 #LT# &3 #AND# &3 #LT# &4 #AND# (( VALUE(&1) +
        VALUE(&2) + VALUE(&3) + VALUE(&4)) #GE# PKSIZE): X4,
OVER4;
ENDSETS
!-----
!Compute the overage of each bundle;
@FOR( BNDL2( I,J):
    OVER2(I,J) = VALUE(I) + VALUE(J) - PKSIZE;
);
@FOR( BNDL3( I,J,K):
    OVER3(I,J,K) = VALUE(I)+VALUE(J)+VALUE(K) - PKSIZE
);
@FOR( BNDL4( I,J,K,L):
    OVER4(I,J,K,L) = VALUE(I)+VALUE(J)+VALUE(K)+VALUE(L) -
PKSIZE;
);

! Maximize score of packages assembled. Penalize a package
that
is over the minimum package size;
MAX= @SUM( BNDL2( I,J): X2(I,J) * (1-(OVER2(I,J)/PKSIZE)^2))
+ @SUM( BNDL3( I,J,K):
            X3(I,J,K) * (1-(OVER3(I,J,K)/PKSIZE)^2))
+ @SUM( BNDL4( I,J,K,L):
            X4(I,J,K,L) * (1-
(OVER4(I,J,K,L)/PKSIZE)^2));

@FOR( OBJECT( M):
! Each object M can be in at most one of the selected bundles;
    @SUM( BNDL2( I, J) | I #EQ# M #OR# J #EQ# M: X2( I, J))
    + @SUM( BNDL3( I, J, K) | I #EQ# M #OR# J #EQ# M #OR# K #EQ#
M:
            X3( I, J, K))
    + @SUM( BNDL4( I, J, K, L) |
            I #EQ# M #OR# J #EQ# M #OR# K #EQ# M #OR# L #EQ# M:
            X4( I, J, K, L)) <= 1;
);

! The X's must = 0 or 1;
@FOR( BNDL2( I, J): @BIN( X2( I, J)););
@FOR( BNDL3( I, J, K): @BIN( X3( I, J, K)););
@FOR( BNDL4( I, J, K, L): @BIN( X4( I, J, K, L)););
END

```

```

Global optimal solution found at iteration: 19
Objective value: 7.989192
      Variable          Value
      X2( A, H) 1.000000
      OVER2( A, H) 5.000000
      X2( B, P) 1.000000
      OVER2( B, P) 63.000000
      X2( C, F) 1.000000
      OVER2( C, F) 60.000000
      X2( D, Q) 1.000000
      OVER2( D, Q) 4.000000
      X2( E, L) 1.000000
      OVER2( E, L) 42.000000
      X2( I, R) 1.000000
      OVER2( I, R) 1.000000
      X2( J, N) 1.000000
      OVER2( J, N) 37.000000
      X3( G, K, M) 1.000000
      OVER3( G, K, M) 8.000000

```

Notice that this allocation is slightly more balanced than the previous solution based on the assignment formulation. The largest “overage” is 63,000 rather than 120,000. This is because the grouping formulation provided an easy way to penalize large packages.

11.7.6 Groups with A Variable Number of Members, Cutting Stock Problem

Another application in which the partitioning or packing approach has worked well is the cutting stock problem in the paper and steel industry. We revisit the example introduced in chapter 7. There we manually enumerated all possible patterns or packings drawn from 8 different finished good widths into each of three different raw material widths. The formulation below automatically enumerates all possible patterns. For each raw material width, the formulation automatically enumerates all possible groupings of 1, 2, ..., 7 finished good widths so that the sum of the finished good widths is less than or equal to the raw material width.

One notable feature of this formulation is that it introduces a shortcut that may be important in keeping computation time low when there are many, e.g., more than 20, objects. To illustrate the shortcut, consider the three declarations:

```

! Enumerate all possible cutting patterns with 1 fg;
rxf(rm,fg)| lenf(&2) #le# lenr(&1): x1;
! Enumerate all possible patterns with 2 fg;
rxf2( rxf, fg) |
    &2 #le# &3 #and# (lenf(&2) + lenf(&3) #le# lenr(&1)): x2;
! Enumerate all possible patterns with 3 fg;
rxf3( rxf2, fg) | &3 #le# &4
    #and# (lenf(&2) + lenf(&3)+ lenf(&4) #le# lenr(&1)): x3;

```

The declaration `rxf(rm, fg)`, by itself, tells LINGO to generate all combinations of one raw material and one finished good. The condition `| lenf(&2) #le# lenr(&1)`, however, tells LINGO to not generate a combination of a raw material(the index &1) and finished good(index &2) for which the length(or width depending upon your orientation) of the finished good is greater than that of the raw material. So, for example, the combination (R36, F38) will not be a member of `rxf`. There will be four elements in `rxf` for which the first item of the pair is R36, namely (R36, F34), (R36, F24), (R36, F15), and (R36, F10).

Now consider how to generate all feasible combinations containing two finished good widths. The obvious declaration would be: `rxf2(rm, fg, fg) | &2 #le# &3 #and# (lenf(&2) + lenf(&3) #le# lenr(&1))`

The condition `&2 #le# &3` says we do not care about the order of the finished goods in the pattern, so we might as well restrict ourselves to listing the finished goods in the pattern in sorted order. The condition `lenf(&2) + lenf(&3) #le# lenr(&1)` restricts the elements of set `rxf2` to feasible ones. This declaration would be valid, but we did not do it. Why? Instead we used the declaration `rxf2(rxf, fg)`. The latter was used mainly for computational reasons. With the latter, LINGO considers every combination of the elements of the set `rxf` and each finished good. Consider the case when the raw material is `r36`. If the declaration `rxf2(rm, fg, fg)` is used, then LINGO would look at $8 * 8 = 64$ combinations of two finished goods and keep only the four combinations (`r36, f24, f10`), (`r36, f15, f15`), (`r36, f15, f10`), and (`r36, f10, f10`). If on the other hand, the declaration `rxf2(rxf, fg)` is used, then when the raw material is `R36`, LINGO will only consider $4 * 8 = 32$ combinations. The 4 arises because set `rxf` contains only 4 elements for which the first member of the pair is `R36`. For sets `rxf3`, and higher, the computational savings can be even higher.

```

! Cutting stock solver(cutgent);
! Keywords: cutting stock;
SETS:
! Each raw material has a size(length) and quantity;
rm: lenr, qr;
! Ditto for each finished good;
fg: lenf, qf;
ENDSETS
DATA:
! Describe the raw materials available;
rm, lenr, qr =
R72 72 9999
R45 48 9999
R36 36 9999;
! Describe the finished goods needed;
fg, lenf, qf =
F60 60 500
F56 56 400
F42 42 300
F38 38 450
F34 34 350
F24 24 100

```

```

F15 15 800
F10 10 1000;
ENDDATA

SETS:
! Enumerate all possible cutting patterns with 1 fg;
rxf(rm,fg)| lenf(&2) #le# lenr(&1): x1;
! Enumerate all possible patterns with 2 fg;
rxf2( rxf, fg) |
&2 #le# &3 #and# (lenf(&2) + lenf(&3) #le# lenr(&1)) :
x2;
! Enumerate all possible patterns with 3 fg;
rxf3( rxf2, fg) | &3 #le# &4 #and#
(lenf(&2) + lenf(&3)+ lenf(&4) #le# lenr(&1)) :
x3;
! Enumerate all possible patterns with 4 fg;
rxf4( rxf3, fg) | &4 #le# &5 #and#
(lenf(&2) + lenf(&3) + lenf(&4)+lenf(&5) #le# lenr(&1)) :
x4;
! Enumerate all possible patterns with 5 fg;
rxf5( rxf4, fg) | &5 #le# &6 #and# (lenf(&2) + lenf(&3) +
lenf(&4)+lenf(&5)+lenf(&6)
#le# lenr(&1)): x5;
! Enumerate all possible patterns with 6 fg;
rxf6( rxf5, fg) | &6 #le# &7 #and# (lenf(&2) + lenf(&3) +
lenf(&4)+lenf(&5)
+lenf(&6)+lenf(&7) #le# lenr(&1)): x6;

ENDSETS
! Minimize length of material used;

MIN = @SUM( rxf(r,f1): lenr(r)*x1(r,f1))
+ @SUM( rxf2(r,f1,f2): lenr(r)*x2(r,f1,f2))
+ @SUM( rxf3(r,f1,f2,f3): lenr(r)*x3(r,f1,f2,f3))
+ @SUM( rxf4(r,f1,f2,f3,f4): lenr(r)*x4(r,f1,f2,f3,f4))
+ @SUM( rxf5(r,f1,f2,f3,f4,f5):
lenr(r)*x5(r,f1,f2,f3,f4,f5))
+ @SUM( rxf6(r,f1,f2,f3,f4,f5,f6):
lenr(r)*x6(r,f1,f2,f3,f4,f5,f6));

! We have to satisfy each finished good demand;
@FOR( fg(f):
@SUM(rxf(r,f): x1(r,f))
+ @SUM(rxf2(r,f1,f2) | f #eq# f1: x2(r,f1,f2))
+ @SUM(rxf2(r,f1,f2) | f #eq# f2: x2(r,f1,f2))
+ @SUM(rxf3(r,f1,f2,f3) | f #eq# f1: x3(r,f1,f2,f3))
+ @SUM(rxf3(r,f1,f2,f3) | f #eq# f2: x3(r,f1,f2,f3))
+ @SUM(rxf3(r,f1,f2,f3) | f #eq# f3: x3(r,f1,f2,f3))

```

```

+ @SUM(rxf4(r,f1,f2,f3,f4) | f #eq# f1: x4(r,f1,f2,f3,f4))
+ @SUM(rxf4(r,f1,f2,f3,f4) | f #eq# f2: x4(r,f1,f2,f3,f4))
+ @SUM(rxf4(r,f1,f2,f3,f4) | f #eq# f3: x4(r,f1,f2,f3,f4))
+ @SUM(rxf4(r,f1,f2,f3,f4) | f #eq# f4: x4(r,f1,f2,f3,f4))
+ @SUM(rxf5(r,f1,f2,f3,f4,f5) | f #eq# f1:
x5(r,f1,f2,f3,f4,f5))
+ @SUM(rxf5(r,f1,f2,f3,f4,f5) | f #eq# f2:
x5(r,f1,f2,f3,f4,f5))
+ @SUM(rxf5(r,f1,f2,f3,f4,f5) | f #eq# f3:
x5(r,f1,f2,f3,f4,f5))
+ @SUM(rxf5(r,f1,f2,f3,f4,f5) | f #eq# f4:
x5(r,f1,f2,f3,f4,f5))
+ @SUM(rxf5(r,f1,f2,f3,f4,f5) | f #eq# f5:
x5(r,f1,f2,f3,f4,f5))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6) | f #eq# f1:
x6(r,f1,f2,f3,f4,f5,f6))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6) | f #eq# f2:
x6(r,f1,f2,f3,f4,f5,f6))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6) | f #eq# f3:
x6(r,f1,f2,f3,f4,f5,f6))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6) | f #eq# f4:
x6(r,f1,f2,f3,f4,f5,f6))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6) | f #eq# f5:
x6(r,f1,f2,f3,f4,f5,f6))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6) | f #eq# f6:
x6(r,f1,f2,f3,f4,f5,f6))
>= qf(f);
);

! We cannot use more raw material than is available;
@FOR( rm( r):
  @SUM(rxf(r,f): x1(r,f))
+ @SUM(rxf2(r,f1,f2): x2(r,f1,f2))
+ @SUM(rxf3(r,f1,f2,f3): x3(r,f1,f2,f3))
+ @SUM(rxf4(r,f1,f2,f3,f4): x4(r,f1,f2,f3,f4))
+ @SUM(rxf5(r,f1,f2,f3,f4,f5): x5(r,f1,f2,f3,f4,f5))
+ @SUM(rxf6(r,f1,f2,f3,f4,f5,f6): x6(r,f1,f2,f3,f4,f5,f6))
  <= qr(r);
);

! Can only run integer quantities of each pattern;
@FOR(rxf: @GIN(x1));
@FOR(rxf2: @GIN(x2));
@FOR(rxf3: @GIN(x3));
@FOR(rxf4: @GIN(x4));
@FOR(rxf5: @GIN(x5));
@FOR(rxf6: @GIN(x6));

```

If you click on LINGO | Generate menu item, to display the scalar version of the model, you can see that the constraint for the 56 inch width is(hopefully reassuringly):

$$X2_R72_F56_F15 + X2_R72_F56_F10 + X1_R72_F56 \geq 400 ;$$

When we click on the Solve icon we get the solution:

```
Global optimal solution found at iteration:      31
Objective value:                      119832.0
```

	Variable	Value
	X2(R72, F60, F10)	500.0000
	X2(R72, F56, F15)	400.0000
	X2(R72, F38, F34)	350.0000
	X3(R72, F42, F15, F15)	186.0000
	X3(R72, F38, F24, F10)	100.0000
	X4(R72, F42, F10, F10, F10)	114.0000
	X4(R45, F15, F10, F10, F10)	2.000000
	X6(R72, F15, F15, F10, F10, F10)	13.00000

11.7.7 Groups with A Variable Number of Members, Vehicle Routing

The following vehicle routing example demonstrates that you can in fact perform a little optimization computation as part of the column or group generation. The example we use is a variation of the vehicle routing problem considered in section 11.6.3. The first and major part of this model is devoted to enumerating all minimal feasible trips with at most seven stops. By feasible trip, we mean that the amount of material to be delivered to the stops in the trip does not exceed the vehicle capacity of 18 pallets. By minimal we mean that for a trip that visits a given set of stops, the trip visits the stops in a sequence that minimizes the distance traveled.

Given that all minimal feasible trips have been generated, the following simple integer program is solved:

Minimize Cost of trips selected;

Subject to:

For each stop:

Exactly one trip includes this stop.

This little example has 15 stops, so the integer program has 15 constraints, and a large number of 0/1 variables(about 7300 in fact), equal in number to the number of minimal feasible trips.

The tricky part is how we generate the sets of minimal feasible trips, PSET2, PSET3, etc. and the associated minimal distances, D2(), D3(), etc. To start understanding ideas, consider the variable D4(I, J, K, L). It has the following definition.

D4(I, J, K, L) = minimum distance required to start at the depot, visit stops I, J, and K in any order and then visit stop L. If DIST(I, J) is the distance matrix, then Held and Karp(1962) observed that if D3 is defined in similar fashion, then D4 can be computed by the dynamic programming recursion:

$$\begin{aligned} D4(I, J, K, L) = \min [& D3(I, J, K) + DIST(K, L), \\ & D3(I, K, J) + DIST(J, L), \\ & D3(J, K, I) + DIST(I, L)] \end{aligned}$$

The complete formulation follows.

```

MODEL: ! (vrgenext);
! The Vehicle Routing Problem (VRP) occurs in many service
systems such as delivery, customer pick-up, repair and
maintenance. A fleet of vehicles, each with fixed
capacity, starts at a common depot and returns to the
depot after visiting locations where service is demanded.
This LINGO model generates all feasible one vehicle routes
and then chooses the least cost feasible multi-vehicle
combination;
SETS:
CITY: Q;
! Q(I) = amount required at city I(given),
must be delivered by just 1 vehicle;
CXC( CITY, CITY): DIST;
! DIST(I,J) = distance from city I to city J;
ENDSETS
DATA:
CITY= Chi Den Frsn Hous KC LA Oakl Anah Peor Phnx Prtl Rvrs Sacr SLC Sntn SBrn;
! Amount to be delivered to each customer;
Q= 0   6   3   7   7   18   4   5   2   6   7   2   4   3   3   3   2 ;
! city 1 represents the common depot, i.e. Q( 1 ) = 0;
! Distance from city I to city J is same(but need not be) from J to I;
DIST= ! To City;
!Chi Den Frsn Hous KC LA Oakl Anah Peor Phnx Prtl Rvrs Sacr SLC Sntn SBrn From;
0 996 2162 1067 499 2054 2134 2050 151 1713 2083 2005 2049 1390 1187 1996 ! Chicago;
996 0 1167 1019 596 1059 1227 1055 904 792 1238 1010 1142 504 939 1001 ! Denver;
2162 1167 0 1747 1723 214 168 250 2070 598 745 268 162 814 1572 265 ! Fresno;
1067 1019 1747 0 710 1538 1904 1528 948 1149 2205 1484 1909 1438 197 1533 ! Huston;
499 596 1723 710 0 1589 1827 1579 354 1214 1809 1535 1742 1086 759 1482 ! K-City;
2054 1059 214 1538 1589 0 371 36 1943 389 959 54 376 715 1363 59 ! L. A.;
2134 1227 168 1904 1827 371 0 407 2043 755 628 425 85 744 1729 422 ! Oaklnd;
2050 1055 250 1528 1579 36 407 0 1933 379 995 45 412 711 1353 55 ! Anahm;
151 904 2070 948 354 1943 2043 1933 0 1568 2022 1889 1958 1299 1066 1887 ! Peoria;
1713 792 598 1149 1214 389 755 379 1568 0 1266 335 760 648 974 333 ! Phnx;
2083 1238 745 2205 1809 959 628 995 2022 1266 0 1001 583 767 2086 992 ! Prtlnd;
2005 1010 268 1484 1535 54 425 45 1889 335 1001 0 430 666 1309 10 ! Rvrsde;
2049 1142 162 1909 1742 376 85 412 1958 760 583 430 0 659 1734 427 ! Scrmto;
1390 504 814 1438 1086 715 744 711 1299 648 767 666 659 0 1319 657 ! SLC;
1187 939 1572 197 759 1363 1729 1353 1066 974 2086 1309 1734 1319 0 1307 ! SAnt;
1996 1001 265 1482 1533 59 422 55 1887 333 992 10 427 657 1307 0 ! SBrn;;
! VCAP is the capacity of a vehicle in 40"x48" pallets;
VCAP = 18;
ENDDATA
SETS:
! Enumerate all sets of various sizes of cities that are load
feasible;
SET2(CITY,CITY)|&1 #GT# 1 #AND# &1 #LT# &2
#AND# (Q(&1)+Q(&2)#LE# VCAP):;
SET3(SET2,CITY)|&2 #LT# &3
#AND# (Q(&1)+Q(&2)+Q(&3)#LE# VCAP):;
SET4(SET3,CITY)|&3 #LT# &4
#AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)#LE# VCAP):;
SET5(SET4,CITY)|&4 #LT# &5
#AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)+Q(&5)#LE# VCAP):;
```

```

SET6(SET5,CITY) | &5 #LT# &6
    #AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)+Q(&5)+Q(&6) #LE# VCAP) :;
SET7(SET6,CITY) | &6 #LT# &7
    #AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)+Q(&5)+Q(&6)+Q(&7) #LE# VCAP) :;

! Enumerate all partially ordered sets with a
specific city as the last one;
PSET2(CITY,CITY) | &1 #GT# 1 #AND# &1#NE#&2
    #AND# (Q(&1)+Q(&2) #LE# VCAP) : D2,X2;
PSET3(SET2,CITY) | &1#NE#&3 #AND# &2#NE#&3
    #AND# (Q(&1)+Q(&2)+Q(&3) #LE# VCAP) : D3,X3;
PSET4(SET3,CITY) | &1#NE#&4 #AND# &2#NE#&4 #AND# &3 #NE# &4
    #AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4) #LE# VCAP) : D4,X4;
PSET5(SET4,CITY) | &1#NE#&5 #AND# &2#NE#&5 #AND# &3 #NE# &5
    #AND# &4 #NE# &5
    #AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)+Q(&5) #LE# VCAP) : D5,X5;
PSET6(SET5,CITY) | &1#NE#&6 #AND#
    &2#NE#&6 #AND# &3 #NE# &6 #AND# &4 #NE# &6 #AND# &5 #NE# &6
    #AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)+Q(&5)+Q(&6) #LE# VCAP) : D6,X6;
PSET7(SET6,CITY) | &1#NE#&7 #AND# &2#NE#&7 #AND# &3#NE#&7 #AND#
    &4#NE#&7 #AND# &5#NE#&7 #AND# &6#NE#&7
    #AND# (Q(&1)+Q(&2)+Q(&3)+Q(&4)+Q(&5)+Q(&6)+Q(&7) #LE# VCAP) : D7,X7;
ENDSETS

! Compute shortest distance to visit all cities in PSET, and
ending up at last city in each
partially ordered set, using Held&Karp DP.
Essential idea:
DS( S,t ) = minimum distance to visit all cities in
S and then end up at t. The recursion is:
DS( S, t ) = min{ k in S: DS(S-k,k) + DIST(k,t) };

@FOR(PSET2(I,J):
    D2(I,J) = DIST(1,I) + DIST(I,J);
    @BIN(X2);
);

@FOR(PSET3(I,J,K):
    ! @SMIN is the min of a list of scalers. D3(I,J,K) = min cost of
    starting at 1, visiting I and J in some order, and then K;
    D3(I,J,K) = @SMIN( D2(I,J) + DIST(J,K), D2(J,I) + DIST(I,K));
    @BIN(X3);
);

@FOR(PSET4(I,J,K,L):
    !D4(I,J,K,L) = min cost of starting at 1, visiting I, J, & K
    in some order, and then L;
    D4(I,J,K,L) =
        @SMIN( D3(I,J,K)+DIST(K,L),
                D3(I,K,J)+DIST(J,L),
                D3(J,K,I)+DIST(I,L));
    @BIN( X4 );
);

@FOR(PSET5(I,J,K,L,M):
    D5(I,J,K,L,M) =

```

```

@SMIN( D4(I,J,K,L)+DIST(L,M),
        D4(I,J,L,K)+DIST(K,M),
        D4(I,K,L,J)+DIST(J,M),
        D4(J,K,L,I)+DIST(I,M));
@BIN(X5);
);

@FOR(PSET6(I,J,K,L,M,N):
D6(I,J,K,L,M,N) =
@SMIN( D5(I,J,K,L,M)+DIST(M,N),
        D5(I,J,K,M,L)+DIST(L,N),
        D5(I,J,L,M,K)+DIST(K,N),
        D5(I,K,L,M,J)+DIST(J,N),
        D5(J,K,L,M,I)+DIST(I,N));
@BIN(X6);
);

@FOR(PSET7(I,J,K,L,M,N,P):
D7(I,J,K,L,M,N,P) =
@SMIN( D6(I,J,K,L,M,N)+DIST(N,P),
        D6(I,J,K,L,N,M)+DIST(M,P),
        D6(I,J,K,M,N,L)+DIST(L,P),
        D6(I,J,L,M,N,K)+DIST(K,P),
        D6(I,K,L,M,N,J)+DIST(J,P),
        D6(J,K,L,M,N,I)+DIST(I,P));
@BIN(X7);
);

! and finally, the optimization model...
Min cost of routes chosen, over complete routes ending back at 1;
Min =
+ @SUM( PSET2(I,J) | J #EQ# 1: D2(I,J)*X2(I,J))
+ @SUM( PSET3(I,J,K) | K #EQ# 1: D3(I,J,K)*X3(I,J,K))
+ @SUM( PSET4(I,J,K,L) | L #EQ# 1: D4(I,J,K,L)*X4(I,J,K,L))
+ @SUM( PSET5(I,J,K,L,M) | M #EQ# 1: D5(I,J,K,L,M)*X5(I,J,K,L,M))
+ @SUM( PSET6(I,J,K,L,M,N) | N #EQ# 1:
                    D6(I,J,K,L,M,N)*X6(I,J,K,L,M,N))
+ @SUM( PSET7(I,J,K,L,M,N,P) | P #EQ# 1:
                    D7(I,J,K,L,M,N,P)*X7(I,J,K,L,M,N));
;

! Each city must be on exactly one complete route;
@FOR( CITY(I1) | I1 #GT# 1:
+ @SUM( PSET2(I,J) | J #EQ# 1 #AND# ( I #EQ# I1): X2(I,J))
+ @SUM( PSET3(I,J,K) | K #EQ# 1 #AND# ( I#EQ#I1 #OR# J#EQ# I1):
                    X3(I,J,K))
+ @SUM( PSET4(I,J,K,L) | L #EQ# 1 #AND#
                    (I#EQ#I1 #OR# J#EQ# I1 #OR# K#EQ# I1): X4(I,J,K,L))
+ @SUM( PSET5(I,J,K,L,M) | M #EQ# 1 #AND#
                    (I#EQ#I1 #OR# J#EQ# I1 #OR# K#EQ# I1 #OR# L#EQ#I1):
                    X5(I,J,K,L,M))
+ @SUM( PSET6(I,J,K,L,M,N) | N #EQ# 1 #AND#
                    (I#EQ#I1 #OR# J#EQ# I1 #OR# K#EQ# I1 #OR# L#EQ#I1
                    #OR# M#EQ#I1): X6(I,J,K,L,M,N))
;
```

```

+ @SUM( PSET7(I,J,K,L,M,N,P) | P #EQ# 1 #AND#
      (I#EQ#I1 #OR# J#EQ#I1 #OR# K#EQ#I1 #OR# L#EQ#I1
       #OR# M#EQ#I1 #OR# N#EQ#I1)
      : X7(I,J,K,L,M,N,P) ) = 1;
);

```

It takes about 4 seconds to get the following solution

Global optimal solution found at iteration:	134
Objective value:	17586.00
Variable	Value
X2(LA, CHI)	1.000000
X3(KC, PEOR, CHI)	1.000000
X4(DEN, HOUS, SNTN, CHI)	1.000000
X5(FRSN, OAKL, PRTL, SACR, CHI)	1.000000
X6(ANAH, PHNX, RVRS, SLC, SBRN, CHI)	1.000000

The obvious question is, how well does this formulation scale up? The final set partitioning integer program is not too challenging. The big challenge is generating and storing the possibly huge number of trips. A crucial consideration is the number of stops per trip. If this is small, e.g., three, then the number of trips will be manageable. A typical vehicle routing problem may have around 100 stops. The number of possible minimal distance trips, each of which visit three out of 100 stops, is $100!/[3!97!] = 161,700$. This is a manageable number of variables for an efficient IP solver.

11.8 Linearizing Products of Variables

We have previously seen products of 0/1 variables, such as $y_1 \times y_2$ and y_1^2 can be represented by linear expressions by means of a simple transformation. This transformation generalizes to the case of the product of a 0/1 variable and a continuous variable.

To illustrate, suppose the product $x \times y$ appears in a model, where y is 0/1 while x is nonnegative continuous. We want to replace this nonlinear component by a (somewhat bigger) linear component. If we have an upper bound (M_x) on the value of x , an upper bound (M_y) on the product $x \times y$, and we define $P = x \times y$, then the following linear constraints will cause P to take on the correct value:

$$\begin{aligned} P &\leq x \\ P &\leq M_y \times y \\ P &\geq x - M_x \times (1 - y) \end{aligned}$$

Hanson and Martin (1990) show how this approach is useful in setting prices for products when we allow bundling of products. Bundle pricing is a form of quantity discounting. Examples of products that might be bundled are (a) airfare, hotel, rental car, tours, and meals or (b) computer, monitor, printer, and hard disk. Stigler (1963) showed how a movie distributor might improve profits by leasing bundles of movies rather than leasing individual movies. Bundling assumes it is easy for the seller to assemble the bundle and difficult for a buyer to unbundle. Otherwise, a reseller could buy the bundle at a discount and then sell the individual components at a markup.

11.8.1 Example: Bundling of Products

Microland Software has recently acquired an excellent word processing product to complement its own recently developed spreadsheet product. Microland is contemplating offering the combination of the two products for a discount. After demonstrating the products at a number of diverse professional meetings, Microland developed the following characterization of the market:

Market Segment	Size in 10,000	Maximum Price Market Segment is Willing To Pay for Various Bundles		
		Spreadsheet Only	Wordprocessor Only	Both
Business/Scientific	7	450	110	530
Legal/Administrative	5	75	430	480
Educational	6	290	250	410
Home	4.5	220	380	390

We will refer to each market segment as simply a “customer”. Economic theory suggests a customer will buy the product that gives the greatest consumer surplus, where consumer surplus is defined as the price the customer is willing to pay for the product (the “reservation price”) minus the market price for the product. For example, if the prices for the three bundles, spreadsheet only, word processor only, and both together, were set respectively at 400, 150, and 500, then the business/scientific market would buy the spreadsheet alone because the consumer surplus is 50 vs. -40 and 30 for the other two bundles.

To give a general model of this situation, define:

R_{ij} = reservation price of customer i for bundle j ,

N_i = “size of customer” i (i.e., number of individual customers in segment i),

s_i = consumer surplus achieved by customer i ,

y_{ij} = 1 if customer i buys bundle j , 0 otherwise,

x_j = price of bundle j set by the seller.

We will treat the empty bundle as just another bundle, so we can say every customer buys exactly one bundle.

The seller, Microland, would like to choose the x_j to:

$$\text{Maximize } \sum_i \sum_j N_i y_{ij} x_j$$

The fact that each customer will buy exactly one bundle is enforced with:

For each customer i :

$$\sum_j y_{ij} = 1$$

For each customer i , its achieved consumer surplus is:

$$s_i = \sum_j (R_{ij} - x_j) y_{ij}$$

Customer i will buy only the bundle j for which its consumer surplus, s_i , is the maximum. This is enforced by the constraints:

For each customer i and bundle j :

$$s_i \geq R_{ij} - x_j$$

A difficulty with the objective function and the consumer surplus constraints is they involve the product $y_{ij}x_j$. Let us follow our previous little example and replace the product $y_{ij}x_j$ by P_{ij} . If M_j is an upper bound on x_j , then, proceeding as before, to enforce the definition $P_{ij} = y_{ij}x_j$, we need the constraints:

$$\begin{aligned} P_{ij} &\leq x_j \\ P_{ij} &\leq R_{ij}y_{ij} \\ P_{ij} &\geq x_j - (1 - y_{ij})M_j. \end{aligned}$$

Making these adjustments to the model, we get:

$$\text{Maximize } \sum_i \sum_j N_i P_{ij}$$

subject to:

For each customer i

$$\sum_j y_{ij} = 1;$$

For each customer i , bundle j :

$$s_i \geq R_{ij} - x_j;$$

For each customer i :

$$s_i = \sum_j (R_{ij}y_{ij} - P_{ij});$$

To enforce the nonlinear condition $P_{ij} = y_{ij}x_j$, we have for each i and j :

$$\begin{aligned} P_{ij} &\leq x_j \\ P_{ij} &\leq R_{ij}y_{ij} \\ P_{ij} &\geq x_j - (1 - y_{ij})M_j. \end{aligned}$$

For all i and j :

$$y_{ij} = 0 \text{ or } 1$$

In explicit form, the LINGO model is:

```
MODEL:
SETS:
  MARKET/B, L, E, H/:S, N;
  ITEM/NONE, SO, WO, BOTH/:X;
  MXI(MARKET, ITEM):R, Y, P;
ENDSETS
DATA:
  N = 7, 5, 6, 4.5; ! Market size;
  R = 0 450 110 530 ! Reservation;
    0 75 430 480 !   prices;
    0 290 250 410
    0 220 380 390;
```

```

M = 600; !Max price of any bundle;
ENDDATA
! Maximize our total revenue = price * market size.
P(i,j) = price customer i pays for product or item j,
        if i buys j, else = 0;
MAX = @SUM(MXI(I, J): P(I, J) * N(I));
!Make the pick variables 0/1;
@FOR(MXI:@BIN(Y));
!Each customer or market i picks or buys exactly one bundle;
@FOR(MARKET(I): @SUM(ITEM(J): Y(I, J)) = 1);
! Each customer i's achieved surplus, S(i), must be at
least as good as from every possible bundle;
@FOR(ITEM(I): @FOR(MARKET(J):
    S(I) >= R(I,J) - X(J)));
!Customer i's achieved surplus = reservations price
of item purchased - its price;
@FOR(MARKET(I):
    S(I) = @SUM(ITEM(J): R(I, J) * Y(I, J) - P(I, J))
    );
! Each price variable Pij must be... ;
!   <= Xj , (the published price);
!   <= Rij * Yij (less than reservation price if bought);
!   >= Xj - M + M * Yij ;
@FOR(MXI(I, J): P(I, J) <= X(J);
    P(I, J) <= Y(I, J) * R(I, J);
    R(I, J) >= X(J) - M + M * Y(I, J););
! Price of bundle should be <= sum of component prices;
X(@INDEX(BOTH)) <= X(@INDEX(SO)) + X(@INDEX(WO));
! Price of bundle should be >= any one component;
X(@INDEX(BOTH)) >= X(@INDEX(SO)); X(@INDEX(BOTH)) >= X(@INDEX(WO));
END

```

For the Microland problem, the solution is to set the following prices:

	Spreadsheet Only	Word Processing Only	Both
Bundle Price:	410	380	410

Thus, the business, legal and educational markets will buy the bundle of both products. The home market will buy only the word processor. Total revenues obtained by Microland are 90,900,000. The interested reader may show that, if bundling is not possible, then the highest revenue that Microland can achieve is only 67,150,000.

11.9 Representing Logical Conditions

For some applications, it may be convenient, perhaps even logical, to state requirements using logical expressions. A logical variable can take on only the values TRUE or FALSE. Likewise, a logical expression involving logical variables can take on only the values TRUE or FALSE. There are two major logical operators, #AND# and #OR#, that are useful in logical expressions.

The logical expression:

$$A \#AND\# B$$

is TRUE if and only if both A and B are true.

The logical expression:

$$A \#OR\# B$$

is TRUE if and only if at least one of A and B is true.

It is sometimes useful also to have the logical operator implication (\Rightarrow) written as follows:

$$A \Rightarrow B$$

with the meaning that if A is true, then B must be true.

Logical variables are trivially representable by binary variables with:

TRUE being represented by 1, and

FALSE being represented by 0.

If A , B , and C are 0/1 variables, then the following constraint combinations can be used to represent the various fundamental logical expressions:

Logical Expression	Mathematical Constraints
$C = A \#AND\# B$	$C \leq A$ $C \leq B$ $C \geq A + B - 1$
$C = A \#OR\# B$	$C \geq A$ $C \geq B$ $C \leq A + B$
$A \Rightarrow C$	$A \leq C$

11.10 Problems

- The following problem is known as a segregated storage problem. A feed processor has various amounts of four different commodities, which must be stored in seven different silos. Each silo can contain at most one commodity. Associated with each commodity and silo combination is a loading cost. Each silo has a finite capacity, so some commodities may have to be split over several silos. For a similar problem arising in the loading of fuel tank trucks at Mobil Oil Company, see Brown, Ellis, Graves, and Ronen (1987). The following table contains the data for this problem.

Commodity	Loading Cost per Ton							Amount of Commodity To Be Stored	
	Silo								
	1	2	3	4	5	6	7		
A	\$1	\$2	\$2	\$3	\$4	\$5	\$5	75 tons	
B	2	3	3	3	1	5	5	50 tons	
C	4	4	3	2	1	5	5	25 tons	
D	1	1	2	2	3	5	5	80 tons	
Silo Capacity in Tons	25	25	40	60	80	100	100		

- a) Present a formulation for solving this class of problems.
 - b) Find the minimum cost solution for this particular example.
 - c) How would your formulation change if additionally there was a fixed cost associated with each silo that is incurred if anything is stored in the silo?
2. You are the scheduling coordinator for a small, growing airline. You must schedule exactly one flight out of Chicago to each of the following cities: Atlanta, Los Angeles, New York, and Peoria. The available departure slots are 8 A.M., 10 A.M., and 12 noon. Your airline has only two departure lounges, so at most two flights can be scheduled per slot. Demand data suggest the following expected profit contribution per flight as a function of departure time:

Expected Profit Contribution in \$1000's

Destination	Time		
	8	10	12
Atlanta	10	9	8.5
Los Angeles	11	10.5	9.5
New York	17	16	15
Peoria	6.4	2.5	-1

Formulate a model for solving this problem.

3. A problem faced by an electrical utility each day is that of deciding which generators to start up at which hour based on the forecast demand for electricity each hour. This problem is also known as the unit commitment problem. The utility in question has three generators with the following characteristics:

Generator	Fixed Startup Cost	Fixed Cost per Period of Operation	Cost per Period per Megawatt Used	Maximum Capacity in Megawatts Each Period
A	3000	700	5	2100
B	2000	800	4	1800
C	1000	900	7	3000

There are two periods in a day and the number of megawatts needed in the first period is 2900. The second period requires 3900 megawatts. A generator started in the first period may be used in the second period without incurring an additional startup cost. All major generators (e.g., A, B, and C above) are turned off at the end of each day.

- a) First, assume fixed costs are zero and thus can be disregarded. What are the decision variables?
- b) Give the LP formulation for the case where fixed costs are zero.
- c) Now, take into account the fixed costs. What are the additional (zero/one) variables to define?
- d) What additional terms should be added to the objective function? What additional constraints should be added?

4. *Crude Integer Programming.* Recently, the U.S. Government began to sell crude oil from its Naval Petroleum Reserve in sealed bid auctions. There are typically six commodities or products to be sold in the auction, corresponding to the crude oil at the six major production and shipping points. A “bid package” from a potential buyer consists of (a) a number indicating an upper limit on how many barrels (bbl.) the buyer is willing to buy overall in this auction and (b) any number of “product bids”. Each product bid consists of a product name and three numbers representing, respectively, the bid price per barrel of this product, the minimum acceptable quantity of this product at this price, and the maximum acceptable quantity of this product at this price. Not all product bids of a buyer need be successful. The government usually places an arbitrary upper limit (e.g., 20%) on the percentage of the total number of barrels over all six products one firm is allowed to purchase.

To illustrate the principal ideas, let us simplify slightly and suppose there are only two supply sources/products, which are denoted by A and B . There are 17,000 bbls. available at A while B has 13,000. Also, there are only two bidders, the Mobon and the Exxil companies. The government arbitrarily decides either one can purchase at most 65% of the total available crude. The two bid packages are as follows:

Mobon:

Maximum desired = 16,000 bbls. total.

Product	Bid per Barrel	Minimum Barrels Accepted	Maximum Barrels Wanted
A	43	9000	16,000
B	51	6000	12,000

Exxil:

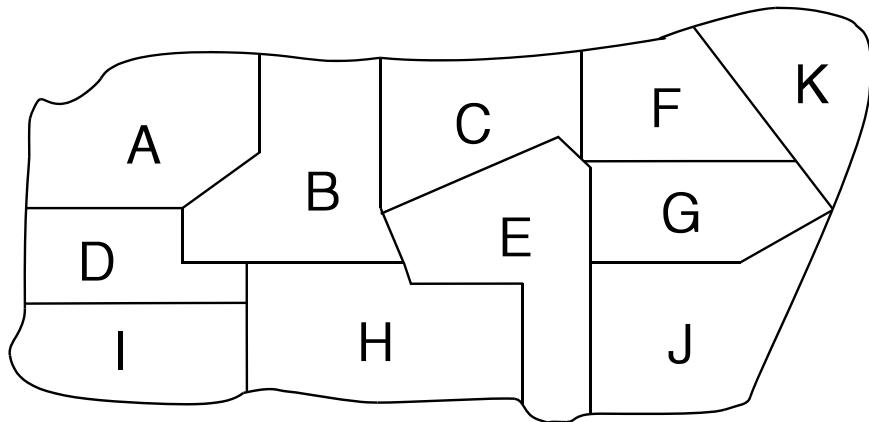
Maximum desired = No limit.

Product	Bid per Barrel	Minimum Barrels Accepted	Maximum Barrels Wanted
A	47	5000	10,000
B	50	5000	10,000

Formulate and solve an appropriate IP for the seller.

5. A certain state allows a restricted form of branch banking. Specifically, a bank can do business in county i if the bank has a “principal place of business” in county i or in a county sharing a nonzero-length border with county i . Figure 11.10 is a map of the state in question:

Figure 11.10 Districts in a State



Formulate the problem of locating a minimum number of principal places of business in the state, so a bank can do business in every county in the state. If the problem is formulated as a covering problem, how many rows and columns will it have? What is an optimal solution? Which formulation is tighter: set covering or simple plant location?

6. *Data Set Allocation Problem.* There are 10 datasets or files, each of which is to be allocated to 1 of 3 identical disk storage devices. A disk storage device has 885 cylinders of capacity. Within a storage device, a dataset will be assigned to a contiguous set of cylinders. Dataset sizes and interactions between datasets are shown in the table below. Two datasets with high interaction rates should not be assigned to the same device. For example, if datasets C and E are assigned to the same disk, then an interaction cost of 46 is incurred. If they are assigned to different disks, there is no interaction cost between C and E .

Dataset for Interaction (Seek Transition) Rates

	A	B	C	D	E	F	G	H	I	J	Dataset Size in Cylinders
A											110
B	43										238
C	120	10									425
D	57	111	188								338
E	96	78	46	88							55
F	83	58	421	60	63						391
G	77	198	207	109	73	74					267
H	31	50	43	47	51	21	88				105
I	38	69	55	21	36	391	47	96			256
J	212	91	84	53	71	40	37	35	221		64
											2249

Find an assignment of datasets to disks, so total interaction cost is minimized and no disk capacity is exceeded.

7. The game or puzzle of mastermind pits two players, a “coder” and a “decoder”, against each other. The game is played with a pegboard and a large number of colored pegs. The pegboard has an array of 4×12 holes. For our purposes, we assume there are only six colors: red, blue, clear, purple, gold, and green. Each peg has only one color. The coder starts the game by selecting four pegs and arranging them in a fixed order, all out of sight of the decoder. This ordering remains fixed throughout the game and is appropriately called the code. At each play of the game, the decoder tries to match the coder’s ordering by placing four pegs in a row on the board. The coder then provides two pieces of information about how close the decoder’s latest guess is to the coder’s order:
- 1) The number of pegs in the correct position (i.e., color matching the coder’s peg in that position), and
 - 2) The maximum number of pegs that would be in correct position if the decoder were allowed to permute the ordering of the decoder’s latest guess.

Call these two numbers m and n . The object of the decoder is to discover the code in a minimum number of plays.

The decoder may find the following IP of interest.

```

MAX = XRED1;
XRED1 + XBLUE1 + XCLEAR1 + XPURP1 + XGOLD1
+ XGREEN1 = 1;
XRED2 + XBLUE2 + XCLEAR2 + XPURP2 + XGOLD2
+ XGREEN2 = 1;
XRED3 + XBLUE3 + XCLEAR3 + XPURP3 + XGOLD3
+ XGREEN3 = 1;
XRED4 + XBLUE4 + XCLEAR4 + XPURP4 + XGOLD4
+ XGREEN4 = 1;
XRED1 + XRED2 + XRED3 + XRED4 - RED = 0;
XBLUE1 + XBLUE2 + XBLUE3 + XBLUE4 - BLUE = 0;
XCLEAR1 + XCLEAR2 + XCLEAR3 + XCLEAR4 - CLEAR = 0;
XPURP1 + XPURP2 + XPURP3 + XPURP4 - PURP = 0;
XGOLD1 + XGOLD2 + XGOLD3 + XGOLD4 - GOLD = 0;
XGREEN1 + XGREEN2 + XGREEN3 + XGREEN4 - GREEN = 0;
END

```

All variables are required to be integer. The interpretation of the variables is as follows. $XRED1 = 1$ if a red peg is in position 1, otherwise 0, etc.; $XGREEN4 = 1$ if a green peg is in position 4, otherwise 0. Rows 2 through 5 enforce the requirement that exactly one peg be placed in each position. Rows 6 through 11 are simply accounting constraints, which count the number of pegs of each color. For example, $RED =$ the number of red pegs in any position 1 through 4. The objective is unimportant. All variables are (implicitly) required to be nonnegative.

At each play of the game, the decoder can add new constraints to this IP to record the information gained. Any feasible solution to the current formulation is a reasonable guess for the next play. An interesting question is what constraints can be added at each play.

To illustrate, suppose the decoder guesses the solution $XBLUE1 = XBLUE2 = XBLUE3 = XRED4 = 1$, and the coder responds with the information that $m = 1$ and $m - n = 1$. That is, one peg is in the correct position and, if permutations were allowed, at most two pegs would be in the correct position. What constraints can be added to the IP to incorporate the new information?

8. The Mathematical Football League (MFL) is composed of M teams (M is even). In a season of $2(M - 1)$ consecutive Sundays, each team will play $(2M - 1)$ games. Each team must play each other team twice, once at home and once at the other team's home stadium. Each Sunday, k games from the MFL are televised. We are given a matrix $\{v_{ij}\}$ where v_{ij} is the viewing audience on a given Sunday if a game between teams i and j playing at team j 's stadium is televised.
 - a) Formulate a model for generating a schedule for the MFL that maximizes the viewing audience over the entire season. Assume viewing audiences are additive.
 - b) Are some values of k easier to accommodate than others? How?

9. The typical automobile has close to two dozen electric motors. However, if you examine these motors, you will see that only about a half dozen distinct motor types are used. For inventory and maintenance reasons, the automobile manufacturer would like to use as few distinct types as possible. For cost, quality, and weight reasons, one would like to use as many distinct motor types as possible, so the most appropriate motor can be applied to each application. The table below describes the design possibilities for a certain automobile:

Application	Number Required	24-Month Failure Probability				
		A	B	C	D	E
Head lamps	2	0.002	0.01		0.01	0.007
Radiator fan	2		0.01	0.002		0.004
Wipers	2				0.007	
Seat	4	0.003			0.006	0.008
Mirrors	2			0.004	0.001	
Heater fan	1		0.006	0.001		
Sun roof	1	0.002			0.003	0.009
Windows	4	0.004	0.008	0.005		
Antenna	1	0.003		0.003	0.002	
Weight		2	3	1.5	1	4
Cost per Motor		24	20	36	28	39

For example, two motors are required to operate the headlamps. If type D motors are used for headlamps, then the estimated probability of a headlamp motor failure in two years is about 0.01. If no entry appears for a particular combination of motor type and application, it means the motor type is inappropriate for that application (e.g., because of size).

Formulate a solvable linear integer program for deciding which motor type to use for each application, so at most 3 motor types are used, the total weight of the motors used is at most 36, total cost of motors used is at most 585, and probability of any failure in two years is approximately minimized.

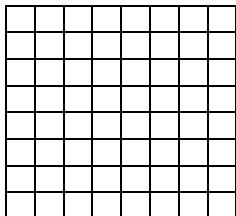
10. We have a rectangular three-dimensional container that is $30 \times 50 \times 50$. We want to pack in it rectangular three-dimensional boxes of the three different sizes: (a) $5 \times 5 \times 10$, (b) $5 \times 10 \times 10$, and (c) $5 \times 15 \times 25$.

A particular packing of boxes into the container is undominated if there is no other packing that contains at least as many of each of the three box types and strictly more of one of the box types.

Show there are no more than 3101 undominated packings.

11. Given the following:

Checkerboard and *domino*



If two opposite corners of the checkerboard are made unavailable, prove there is no way of exactly covering the remaining grid with 31 dominoes.

12. Which of the following requirements could be represented exactly with linear constraints? (You are allowed to use transformations if you wish.)

- (a) $(3 \times x + 4 \times y)/(2 \times x + 3 \times y) \leq 12$;
- (b) $\text{MAX}(x, y) < 8$;
- (c) $3 \times x + 4 \times y \times y \geq 11$; where y is 0 or 1;
- (d) $\text{ABS}(10 - x) \leq 7$ (Note ABS means absolute value);
- (e) $\text{MIN}(x, y) < 12$.

13. A common way of controlling access in many systems, such as information systems or the military, is with priority levels. Each user i is assigned a clearance level U_i . Each object j is assigned a security level L_j . A user i does not have access to object j if the security level of j is higher than the clearance level of i . Given a set of users; and, for each user, a list of objects to which that user does not have access; and a list of objects to which the user should have access, can we assign U_i 's and L_j 's, so these access rights and denials are satisfied? Formulate as an integer program.
14. One of the big consumption items in the U.S. is automotive fuel. Any petroleum distributor who can deliver this fuel reliably and efficiently to the hundreds of filling stations in a typical distribution region has a competitive advantage. This distribution problem is complicated by the fact that a typical customer (i.e., filling station) requires three major products: premium gasoline, an intermediate octane grade (e.g., "Silver"), and regular gasoline. A typical situation is described below. A delivery tank truck has four compartments with capacities in liters of 13,600, 11,200, 10,800, and 4400. We would like to load the truck according to the following limits:

	Liters of		
	Premium	Intermediate	Regular
At least:	8,800	12,000	12,800
At most:	13,200	17,200	16,400

Only one gasoline type can be stored per compartment in the delivery vehicle. Subject to the previous considerations, we would like to maximize the amount of fuel loaded on the truck.

- (a) Define the decision variables you would use in formulating this problem as an IP.
- (b) Give a formulation of this problem.
- (c) What allocation do you recommend?

14

Multiple Criteria and Goal Programming

14.1 Introduction

Until now, we have assumed a single objective or criterion. In reality, however, there may be two or more measures of goodness. Our life becomes more difficult, or at least more interesting, if these multiple criteria are incommensurate (i.e., it is difficult to combine them into a single criterion). The overused phrase for lamenting the difficulty of such situations is “You can’t mix apples and oranges”.

Some examples of incommensurate criteria are:

- risk vs. return on investment,
- short-term profits vs. long-term growth of a firm,
- cost vs. service by a government agency,
- the treatment of different individuals under some policy of an administrative agency (e.g., rural vs. urban citizens, residents near an airport vs. travelers using an airport, and fishermen vs. water transportation companies vs. farmers using irrigation near a large lake).

Multi-criteria situations can be classified into several categories:

1. Criteria are intrinsically different (e.g., risk vs. return, cost vs. service).
 - a) Weights or trade-off rates can be determined;
 - b) Criteria can be strictly ordered by importance. We have so-called preemptive objectives.
2. Criteria are intrinsically similar (i.e., in some sense they should have equal weight).

A rich source of multi-criteria problems is the design and operation of public works. A specific example is the huge “Three Gorges” dam on the Yangtze River in China. Interested parties include: (a) industrial users of electricity, who would like the average water level in the dam to be high, so as to maximize the amount of electricity that can be generated; (b) farmers downstream from the dam, who would like the water level in the dam to be maintained at a low level, so unexpected large rainfalls can be accommodated without overflow and flooding; (c) river shipping interests, who would like the lake level to be allowed to fluctuate as necessary, so as to maintain a steady flow rate out of the dam,

thereby allowing year round river travel by large ships below the dam; (d) lake fishermen and recreational interests, who would like the flow rate out of the dam to be allowed to fluctuate as necessary, so as to maintain a steady lake level; e) irrigation water users who would like the lake level to be high and to be allowed to use water for irrigation than for power generation, and (f) environmental interests, who did not want the dam built in the first place. For the Three Gorges dam in particular, flood control interests have argued for having the water level behind the dam held at 459 feet above sea level just before the rainy season, so as to accommodate storm runoff (see, for example, Fillon (1996)). Electricity generation interests, however, have argued for a water level of 574 feet above sea level to generate more electricity.

14.1.1 Alternate Optima and Multicriteria

If you have a model with alternate optimal solutions, this is nature's way of telling you that you have multiple criteria. You should probably look at your objective function more closely and add more detail. Users do not like alternate optima. If there are alternate optima, the typical solution method will essentially choose among them randomly. If people's jobs or salaries depend upon the "flip of a coin" in your analysis, they are going to be unhappy. Even if careers are not at stake, alternate optima are at least a nuisance. People find it disconcerting if they get different answers (albeit with the same objective value) when they solve the same problem on different computers.

One resolution of alternate optima that might occur to some readers is to take the average of all distinct alternate optima and use this average solution as the final, unique, well-defined answer. Unfortunately, this is usually not practical because:

- a) it may be difficult to enumerate all alternate optima, and
- b) the average solution may be unattractive or even infeasible if the model involves integer variables.

14.2 Approaches to Multi-criteria Problems

There is a variety of approaches to dealing with multiple criteria. Some of the more practical ones are described below.

14.2.1 Pareto Optimal Solutions and Multiple Criteria

A solution to a multi-criteria problem is said to be *Pareto optimal* if there is no other solution that is at least as good according to all criteria and strictly better according to at least one criterion. A Pareto optimal solution is not dominated by any other solution. Clearly, we want to consider only Pareto optimal solutions. If we do not choose our criteria carefully, we might find ourselves recommending solutions that are not Pareto optimal. There are computer programs for multi-criteria linear programming that will generate all the undominated extreme solutions. For a small problem, a decision maker could simply choose the most attractive extreme solution based on subjective criteria. For large problems, the number of undominated extreme solutions may easily exceed 100, so this approach may be overwhelming.

14.2.2 Utility Function Approach

A superficially attractive solution of the multi-criteria problem is the definition of a utility function. If the decision variables are x_1, x_2, \dots, x_n , we "simply" construct the utility function $u(x_1, x_2, \dots, x_n)$ which computes the value or utility of any possible combination of values for the vector x_1, x_2, \dots, x_n . This is a very useful approach for thinking about optimization. However, it has several practical

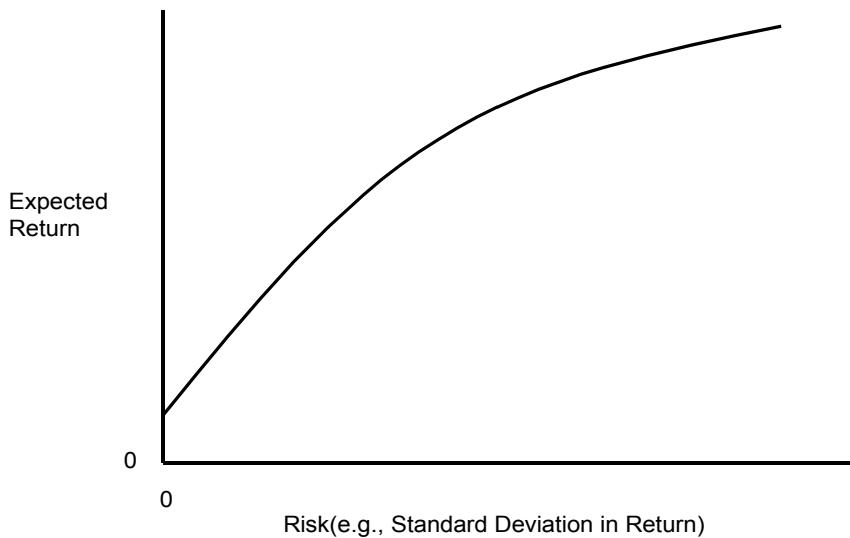
limitations: (a) it may take a lot of work to construct it, and (b) it will probably be highly nonlinear. Feature (b) means we probably cannot use LP to solve the problem.

14.2.3 Trade-off Curves

If we have only two or three criteria, then the trade-off curve approach has most of the attractive features of the utility function approach, but is also fairly practical. We simply construct a curve, the so-called “efficient frontier”, which shows how we can trade off one criterion for another. One of the most well known settings using a trade-off curve is to describe the relationship between two criteria in a financial portfolio. The two criteria are expected return on investment and risk. We want return to be high and risk to be low. Figure 14.1 shows the typical relationship between risk and return. Each point on the curve is Pareto optimal. That is, for any point on the curve, there is no other point with higher expected return and lower risk.

Even though a decision maker has not gone through the trouble of constructing his utility function, he may be able to look at this trade-off curve and perhaps say: “Gee, I am comfortable with an expected return of 8% with standard deviation of 3%.”

Figure 14.1 Trade-off Curve for Risk and Expected Return



14.2.4 Example: Ad Lib Marketing

Ad Lib is a freewheeling advertising agency that wants to solve a so-called media selection problem for one of its clients. It is considering placing ads in five media: late night TV (TVL), prime time TV (TVP), billboards (BLB), newspapers (NEW), and radio (RAD). These ads are intended to reach seven different demographic groups.

The following table gives the number of exposures obtained in each of the seven markets per dollar of advertising in each of five media. The second to last row of the table lists the minimum required number of exposures in each of the seven markets. The feeling is that we must reach this minimum number of readers/viewers, regardless of the cost. The last row of numbers is the saturation level for each market. The feeling is that exposure beyond this level is of no value. Exposures between these two limits will be termed useful exposures.

	Exposure Statistics for Ad Lib Marketing Exposure in 1000's per \$1000 Spent						
	Market Group						
	1	2	3	4	5	6	7
TVL		10	4	50	5		2
TVP		10	30	5	12		
BLB	20					5	3
NEW	8					6	10
RAD		6	5	10	11	4	
Minimum Number of Exposures Needed in 1,000's	25	40	60	120	40	11	15
Saturation Level in 1,000's of Exposures	60	70	120	140	80	25	55

How much money should be spent on advertising in each medium? There are really two criteria: (a) cost (which we want to be low), and (b) useful exposures (which we want to be high). At the outset, we arbitrarily decided we would spend no more than \$11,000.

A useful model can be formulated if we define:

Decision variables:

TVL, TVP, etc. = dollars spent in 1,000's on advertising in *TVL, TVP, etc.*;

UX1, UX2, etc. = number of useful excess exposures obtained in market 1, 2, etc., beyond the minimum (i.e., min {saturation level, actual exposures} – minimum required);

COST = total amount spent on advertising;

USEFULX = total useful exposures.

There will be two main sets of constraints. One set that says:

exposures in a market \geq minimum required + useful excess exposure beyond minimum.

The other says:

useful excess exposures in a market \leq saturation level – minimum required.

An explicit formulation is:

```
[UEXP] MAX = USEFULX ;      ! Maximize useful exposures;
[LIMCOST] COST <= 11; !Limit (in $1,000) on cost;
[LIMEXP]   USEFULX >= 0; ! Required exposures;
[DEFCOST] TVL + TVP + BLB + NEW + RAD = COST;
[DEFEXP] UX1 + UX2 + UX3 + UX4 + UX5 + UX6 + UX7 =
          USEFULX;
[MKT1]           20 * BLB + 8 * NEW           - UX1 >= 25;
[MKT2] 10 * TVL + 10 * TVP           + 6 * RAD - UX2 >= 40;
[MKT3] 4 * TVL + 30 * TVP           + 5 * RAD - UX3 >= 60;
[MKT4] 50 * TVL + 5 * TVP           + 10 * RAD - UX4 >= 120;
[MKT5] 5 * TVL + 12 * TVP           + 11 * RAD - UX5 >= 40;
[MKT6]           5 * BLB + 6 * NEW + 4 * RAD - UX6 >= 11;
[MKT7] 2 * TVL + 3 * BLB + 10 * NEW           - UX7 >= 15;
[RANGE1] UX1 <= 35;
[RANGE2] UX2 <= 30;
[RANGE3] UX3 <= 60;
[RANGE4] UX4 <= 20;
[RANGE5] UX5 <= 40;
[RANGE6] UX6 <= 14;
[RANGE7] UX7 <= 40;
```

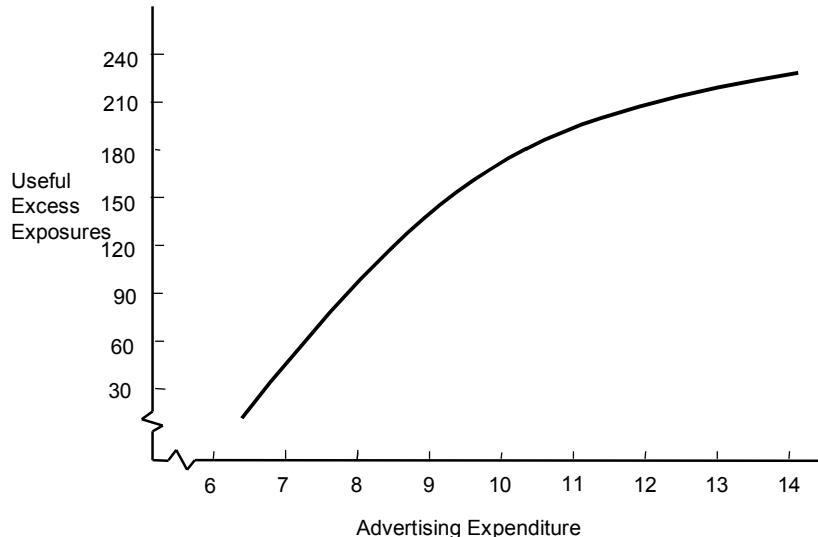
The following is part of the solution to this model:

Optimal solution found at step:	15
Objective value:	196.7626
Variable	Value
USEFULX	196.7626
COST	11.00000
TVL	1.997602
TVP	3.707434
BLB	2.908873
NEW	0.2278177
RAD	2.158273
UX1	35.00000
UX2	30.00000
UX3	60.00000
UX4	20.00000
UX5	38.21823
UX6	13.54436
UX7	0.0000000
Row	Slack or Surplus
UEXP	196.7626
LIMCOST	0.0000000
LIMEXP	196.7626
DEFCOST	0.0000000
DEFEXP	0.0000000

Notice we advertise up to the saturation level in markets 1 to 4. In market 7, we advertise just enough to achieve the minimum required.

If you change the cost limit (initially at 11) to various values ranging from 6 to 14 and plot the maximum possible number of useful exposures, you get a trade-off curve, or efficient frontier, shown in the Figure 14.2:

Figure 14.2 Trade-off Between Exposures and Advertising



14.3 Goal Programming and Soft Constraints

Goal Programming is closely related to the concept of multi-criteria as well as a simple idea that we dub “soft constraints”. Soft constraints and Goal Programming are a response to the following two “laws of the real world”.

In the real world:

- 1) there is always a feasible solution;
- 2) there are no alternate optima.

In practical terms, (1) means a good manager (or one wishing to at least keep a job) never throws up his or her hands in despair and says “no feasible solution”. Law (2) means a typical decision maker will never be indifferent between two proposed courses of action. There are always sufficient criteria to distinguish some course of action as better than all others.

From a model perspective, these two laws mean a well-formulated model (a) always has a feasible solution and (b) does not have alternate optima.

14.3.1 Example: Secondary Criterion to Choose Among Alternate Optima

Here is a standard, seven-day/week staffing problem similar to that discussed in Chapter 7. The variables: M, T, W, R, F, S, N , denote the number of people starting their five-day work week on Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, or Sunday, respectively:

```

MIN = 9*M + 9*T + 9*W + 9*R + 9*F + 9*S + 9*N;
[MON]   M           + R + F + S + N >= 3;
[TUE]   M + T       + F + S + N >= 3;
[WED]   M + T + W  + S + N >= 8;
[THU]   M + T + W + R + N >= 8;
[FRI]   M + T + W + R + F >= 8;
[SAT]   T + W + R + F + S >= 3;
[SUN]   W + R + F + S + N >= 3;
END

```

When solved, we get the following solution:

Optimal solution found at step:	6	
Objective value:	72.00000	
Variable	Value	Reduced Cost
M	5.000000	0.0000000
T	0.0000000	0.0000000
W	3.000000	0.0000000
R	0.0000000	0.0000000
F	0.0000000	9.000000
S	0.0000000	9.000000
N	0.0000000	0.0000000
Row	Slack or Surplus	Dual Price
1	72.00000	1.000000
MON	2.000000	0.0000000
TUE	2.000000	0.0000000
WED	0.0000000	0.0000000
THU	0.0000000	-9.000000
FRI	0.0000000	0.0000000
SAT	0.0000000	0.0000000
SUN	0.0000000	0.0000000

Notice there may be alternate optima (e.g., the slack and dual price in row “WED” are both zero). This solution puts all the surplus capacity on Saturday and Sunday. The different optima might distribute the surplus capacity in different ways over the days of the week. Saturday and Sunday have a lot of excess capacity while the very similar days, Monday and Tuesday, have no surplus capacity.

In terms of multiple criteria, we might say:

- a) our most important criterion is to minimize total staffing cost;
- b) our secondary criterion is to have a little extra capacity, specifically one unit, each day if it will not hurt criterion 1.

To encourage more equitable distribution, we add some “excess” variables (XM , XT , etc.) that give a tiny credit of -1 for each surplus up to at most 1 on each day. The modified formulation is:

```

MODEL:
MIN = 9*M + 9*T + 9*W + 9*R + 9*F + 9*S + 9*N
      - XM - XT - XW - XR - XF - XS - XN;
[MON] M           + R   + F   + S   + N   - XM ≥ 3;
[TUE] M + T       + F   + S   + N   - XT ≥ 3;
[WED] M + T + W   + S   + N   - XW ≥ 8;
[THU] M + T + W + R   + N   - XR ≥ 8;
[FRI] M + T + W + R + F   - XF ≥ 8;
[SAT]   T + W + R + F   + S   - XS ≥ 3;
[SUN]     W + R + F   + S   + N   - XN ≥ 3;
[N9]    XM ≤ 1;
[N10]   XT ≤ 1;
[N11]   XW ≤ 1;
[N12]   XR ≤ 1;
[N13]   XF ≤ 1;
[N14]   XS ≤ 1;
[N15]   XN ≤ 1;
END

```

The solution now is:

Optimal solution found at step:	19	
Objective value:	68.00000	
Variable	Value	Reduced Cost
M	4.000000	0.0000000
T	0.0000000	0.0000000
W	4.000000	0.0000000
R	0.0000000	1.000000
F	0.0000000	8.000000
S	0.0000000	8.000000
N	0.0000000	1.000000
XM	1.000000	0.0000000
XT	1.000000	0.0000000
XW	0.0000000	0.0000000
XR	0.0000000	6.000000
0.0000000	0.0000000	
XS	1.000000	0.0000000
XN	1.000000	0.0000000

Notice, just as before, we still hire a total of eight people, but now the surplus is evenly distributed over the four days M , T , S , and N . This should be a more attractive solution.

14.3.2 Preemptive/Lexico Goal Programming

The above approach required us to choose the proper relative weights for our two objectives, cost and service. In some situations, it may be clear that one objective is orders of magnitude more important than the other. One could choose weights to reflect this (e.g., 99999999 for the first and 0.0000001 for the second), but there are a variety of reasons for not using this approach. First of all, there would probably be numerical problems, especially if there are more than two objectives. A typical computer cannot accurately add numbers that differ by more than 15 orders of magnitude (e.g., 100,000,000 and .0000001).

More importantly, it just seems more straightforward simply to say: "This first objective is far more important than the remaining objectives, the second objective is far more important than the remaining objectives," etc. This approach is sometimes called Preemptive or Lexico goal programming. The following illustrates for our previous staff-scheduling example. The first model solved places a weight of 1.0 on the more important objective, *COST*, and no weight on the secondary objective, *EXTRA* credit for useful overstaffing:

```

!Example of Lexico-goal programming
MIN = 1 * COST - 0 * EXTRA;
[MON] M      + R + F + S + N - XM >= 3;
[TUE] M + T      + F + S + N - XT >= 3;
[WED] M + T + W      + S + N - XW >= 8;
[THU] M + T + W + R      + N - XR >= 8;
[FRI] M + T + W + R + F      - XF >= 8;
[SAT]   T + W + R + F + S      - XS >= 3;
[SUN]   W + R + F + S + N - XN >= 3;
! Upper limit on creditable excess;
[EXM] XM <= 1;
[EXT] XT <= 1;
[EXW] XW <= 1;
[EXR] XR <= 1;
[EXF] XF <= 1;
[EXS] XS <= 1;
[EXN] XN <= 1;
! Define the two objectives;
[OBJCOST] COST = M + R + F + S + N + T + W;
[OBJXTRA] EXTRA = XM + XT + XW + XR + XF + XS + XN;
END

```

The solution is:

Optimal solution found at step:	11	
Objective value:	8.000000	
Variable	Value	Reduced Cost
COST	8.000000	0.0000000
EXTRA	0.0000000	0.0000000
M	3.000000	0.0000000
R	0.0000000	0.0000000
F	0.0000000	0.0000000
S	0.0000000	1.000000
N	0.0000000	1.000000
XM	0.0000000	0.0000000
T	0.0000000	0.0000000
XT	0.0000000	0.0000000
W	5.000000	0.0000000
XW	0.0000000	0.0000000
XR	0.0000000	0.0000000
XF	0.0000000	1.000000
XS	0.0000000	0.0000000
XN	0.0000000	0.0000000
Row	Slack or Surplus	Dual Price
1	8.000000	1.000000
MON	0.0000000	0.0000000
TUE	0.0000000	0.0000000
WED	0.0000000	0.0000000
THU	0.0000000	0.0000000
FRI	0.0000000	-1.000000
SAT	2.000000	0.0000000
SUN	2.000000	0.0000000
EXM	1.000000	0.0000000
EXT	1.000000	0.0000000
EXW	1.000000	0.0000000
EXR	1.000000	0.0000000
EXF	1.000000	0.0000000
EXS	1.000000	0.0000000
EXN	1.000000	0.0000000
OBJCOST	0.0000000	-1.000000
OBJXTRA	0.0000000	0.0000000

Notice because there is zero weight in the objective, it does not claim any *EXTRA* credit for overstaffing by one unit. This solution starts 3 people on Monday, and 5 people on Wednesday. Thus, there is no overstaffing on Monday, Tuesday, Thursday, and Friday, but both Saturday and Sunday are overstaffed by two each. We can try to distribute the overstaffing more evenly by solving the following model. It fixes the *COST* at the minimum we have just learned, and now maximizes (or minimizes the negative of) the creditable extra staffing:

```

MIN = 0 * COST - 1 * EXTRA;
[MON] M + R + F + S + N - XM >= 3;
[TUE] M + T + F + S + N - XT >= 3;
[WED] M + T + W + S + N - XW >= 8;
[THU] M + T + W + R + N - XR >= 8;
[FRI] M + T + W + R + F - XF >= 8;
[SAT] T + W + R + F + S - XS >= 3;
[SUN] W + R + F + S + N - XN >= 3;
! Upper limit on creditable excess;
[EXM] XM <= 1;
[EXT] XT <= 1;
[EXW] XW <= 1;
[EXR] XR <= 1;
[EXF] XF <= 1;
[EXS] XS <= 1;
[EXN] XN <= 1;
! Define the two objectives;
[OBJCOST] COST = M + R + F + S + N + T + W;
[OBJXTRA] EXTRA = XM + XT + XW + XR + XF + XS + XN;
! Fix the cost at its minimum value;
[FXCOST] COST = 8;
END

```

This gives the solution:

Optimal solution found at step:	7
Objective value:	-4.000000
Variable	Value
COST	8.000000
EXTRA	4.000000
M	4.000000
R	0.0000000
F	0.0000000
S	0.0000000
N	0.0000000
XM	1.000000
T	0.0000000
XT	1.000000
W	4.000000
XW	0.0000000
XR	0.0000000
0.0000000	
XS	1.000000
XN	1.000000

Row	Slack or Surplus	Dual Price
1	-4.000000	-1.000000
MON	0.0000000	0.0000000
TUE	0.0000000	0.0000000
WED	0.0000000	-1.000000
THU	0.0000000	-1.000000
FRI	0.0000000	-1.000000
SAT	0.0000000	0.0000000
SUN	0.0000000	0.0000000
EXM	0.0000000	1.000000
EXT	0.0000000	1.000000
EXW	1.000000	0.0000000
EXR	1.000000	0.0000000
EXF	1.000000	0.0000000
EXS	0.0000000	1.000000
EXN	0.0000000	1.000000
OBJCOST	0.0000000	-3.000000
OBJXTRA	0.0000000	1.000000
FXCOST	0.0000000	3.000000

Notice this is a different solution. Nevertheless, still with a cost of 8, but with now an *EXTRA* credit for slight overstaffing of 4. This solution starts 4 people on each of Monday and Wednesday. Thus, Wednesday, Thursday, and Friday have no overstaffing, but Monday, Tuesday, Saturday, and Sunday are overstaffed by one each.

14.4 Minimizing the Maximum Hurt, or Unordered Lexico Minimization

There are some situations in which there are a number of parties that, in some sense, are equal. There may be certain side conditions, however, that prevent us from treating them exactly equally. An example is representation in a House of Representatives. Ideally, we would like to have the number of representatives in a state be exactly proportional to the population of the state. Because the House of Representatives is typically limited to a fixed size and we cannot have fractional representatives (although some voters may feel they have encountered such an anomaly), we will find some states have more citizens per representative than others.

In more general settings, an obvious approach for minimizing such inequities is to choose things, so we minimize the maximum inequity or “hurt.” Once we have minimized the worst hurt, the obvious thing is to minimize the second greatest hurt, etc. We will refer to such a minimization as Unordered Lexico Minimization. For example, if there are four parties, and $(10, 13, 8, 9)$ is the vector of taxes to be paid, then we would say the vector $(13, 8, 9, 9)$ is better in the unordered Lexico-min sense. The highest tax is the same for both solutions, but the second highest tax is lower for the second solution.

Serafini (1996) uses this approach in scheduling jobs in a textile factory in northern Italy. Each job has a due-date. If demand and capacity are such that not all jobs can be completed by their due date, then a reasonable objective is to minimize the maximum lateness of any job. A reasonable sub-objective is to minimize the lateness of the second latest job, etc.

14.4.1 Example

This example is based on one in Sankaran (1989). There are six parties and x_i is the assessment to be paid by party i to satisfy a certain community building project. The x_i must satisfy the set of constraints:

- A. $X1 + 2 X2 + 4 X3 + 7 X4 \geq 16$
- B. $2.5 X1 + 3.5 X2 + 5.2 X5 \geq 17.5$
- C. $0.4 X2 + 1.3 X4 + 7.2 X6 \geq 12$
- D. $2.5 X2 + 3.5 X3 + 5.2 X5 \geq 13.1$
- E. $3.5 X1 + 3.5 X4 + 5.2 X6 \geq 18.2$

We would like to minimize the highest assessment paid by anyone. Given that, we would like to minimize the second highest assessment paid by anyone. Given that, we would like to minimize the third highest, etc. The interested reader may try to improve upon the following set of assessments:

$$X1 = 1.5625$$

$$X2 = 1.5625$$

$$X3 = .305357$$

$$X4 = 1.463362$$

$$X5 = 1.5625$$

$$X6 = 1.463362$$

There is no other solution in which:

- a) the highest assessment is less than 1.5625, and
- b) the second highest assessment is less than 1.5625, and
- c) the third highest assessment is less than 1.5625, and
- d) the fourth highest assessment is less than 1.463362, etc.

14.4.2 Finding a Unique Solution Minimizing the Maximum

A quite general approach to finding a unique unordered Lexico minimum exists when the feasible region is convex (i.e. any solution that is a positively weighted average of two feasible solutions is also feasible). Thus, problems with integer variables are not convex. Let the vector $\{x_1, x_2, \dots, x_n\}$ denote the cost allocated to each of n parties.

If the feasible region is convex, then there is a unique solution and the following algorithm will find it. Maschler, Peleg, and Shapley (1979) discuss this idea in the game theory setting, where the “nucleolus” is a closely related concept. If the feasible region is not convex (e.g., the problem has integer variables), then the following method is not guaranteed to find the solution. Let S be the original set of constraints on the x 's.

- 1) Let $J = \{1, 2, \dots, n\}$, and $k = 0$; (Note: J is the set of parties for whom we do not yet know the final x_i)
- 2) Let $k = k + 1$;

- 3) Solve the problem:

Minimize Z

subject to

x feasible to S and,

$Z \geq x_j$ for j in J

(Note: this finds the minimum, maximum hurt among parties for which we have not yet fixed the x_j 's.);

- 4) Set $Z_k = Z$ of (3), and add to S the constraints:

$x_j \leq Z_k$ for all j in J ;

- 5) Set $L = \{j \in J \text{ for which } x_j = Z_k \text{ in (3)}\}$:

For each j in L :

Solve:

Minimize x_j

subject to

x feasible to S ;

If $x_j = Z_k$, then set $J = J - j$, and append to S the constraint $x_j = Z_k$

- 6) If J is not empty, go to (2), else we are done.

To find the minimum maximum assessment for our example problem, we solve the following problem:

```
MODEL:
MIN = Z;
! The physical constraints on the X's;
[A] X1 + 2*X2 + 4*X3 + 7*X4 >= 16;
[B] 2.5*X1 + 3.5*X2 + 5.2*X5 >= 17.5;
[C] 0.4*X2 + 1.3*X4 + 7.2*X6 >= 12;
[D] 2.5*X2 + 3.5*X3 + 5.2*X5 >= 13.1;
[E] 3.5*X1 + 3.5*X4 + 5.2*X6 >= 18.2;
! Constraints to compute the max hurt Z;
[H1] Z - X1 >= 0;
[H2] Z - X2 >= 0;
[H3] Z - X3 >= 0;
[H4] Z - X4 >= 0;
[H5] Z - X5 >= 0;
[H6] Z - X6 >= 0;
END
```

Its solution is:

Objective value:	1.5625000	
Variable	Value	Reduced Cost
Z	1.5625000	0.0000000
X1	1.5625000	0.0000000
X2	1.5625000	0.0000000
X3	1.5625000	0.0000000
X4	1.5625000	0.0000000
X5	1.5625000	0.0000000
X6	1.5625000	0.0000000

Thus, at least one party will have a “hurt” of 1.5625. Which party or parties will it be?

Because all six x_i 's equal 1.5625, we solve a series of six problems such as the following:

```

MODEL:
MIN = X1;
! The physical constraints on the X's;
[A] X1 + 2*X2 + 4*X3 + 7*X4 >= 16;
[B] 2.5*X1 + 3.5*X2 + 5.2*X5 >= 17.5;
[C] 0.4*X2 + 1.3*X4 + 7.2*X6 >= 12;
[D] 2.5*X2 + 3.5*X3 + 5.2*X5 >= 13.1;
[E] 3.5*X1 + 3.5*X4 + 5.2*X6 >= 18.2;
! Constraints for finding the minmax hurt, Z;
[H1] X1 <= 1.5625000;
[H2] X2 <= 1.5625000;
[H3] X3 <= 1.5625000;
[H4] X4 <= 1.5625000;
[H5] X5 <= 1.5625000;
[H6] X6 <= 1.5625000;
END

```

The solution for the case of $X1$ is:

Objective value:	1.5625000	
Variable	Value	Reduced Cost
X1	1.5625000	0.0000000
X2	1.5625000	0.0000000
X3	0.3053573	0.0000000
X4	1.5625000	0.0000000
X5	1.5625000	0.0000000
X6	1.3966350	0.0000000

Thus, there is no solution with all the x_i 's ≤ 1.5625 , but with $X1$ strictly less than 1.5625. So, we can fix $X1$ at 1.5625. Similar observations turn out to be true for $X2$ and $X5$.

So, now we wish to solve the following problem:

```

MODEL:
MIN = Z;
! The physical constraints on the X's;
X1 + 2*X2 + 4*X3 + 7*X4 >= 16;
2.5*X1 + 3.5*X2 + 5.2*X5 >= 17.5;
0.4*X2 + 1.3*X4 + 7.2*X6 >= 12;
2.5*X2 + 3.5*X3 + 5.2*X5 >= 13.1;
3.5*X1 + 3.5*X4 + 5.2*X6 >= 18.2;
! Constraints for finding the minmax hurt, Z;
X1 = 1.5625000;
X2 = 1.5625000;
- Z + X3 <= 0;
- Z + X4 <= 0;
X5 = 1.5625000;
- Z + X6 <= 0;
END

```

Upon solution, we see the second highest “hurt” is 1.4633621:

Objective value:	1.4633621	
Variable	Value	Reduced Cost
Z	1.4633621	0.0000000
X1	1.5625000	0.0000000
X2	1.5625000	0.0000000
X3	1.4633621	0.0000000
X4	1.4633621	0.0000000
X5	1.5625000	0.0000000
X6	1.4633621	0.0000000

Any or all of X_3 , X_4 or X_6 could be at this value in the final solution. Which ones? To find out, we solve the following kind of problem for X_3 , X_4 and X_6 :

```

MODEL:
MIN = X3;
! The physical constraints on the X's;
[A]   X1 + 2*X2 + 4*X3 + 7*X4    >=      16;
[B]   2.5*X1 + 3.5*X2 + 5.2*X5    >=     17.5;
[C]   0.4*X2 + 1.3*X4 + 7.2*X6    >=      12;
[D]   2.5*X2 + 3.5*X3 + 5.2*X5    >=     13.1;
[E]   3.5*X1 + 3.5*X4 + 5.2*X6    >=     18.2;
! Constraints for finding the minmax hurt, Z;
[H1]   X1 = 1.5625000;
[H2]   X2 = 1.5625000;
[H3]   X3 <= 1.4633621;
[H4]   X4 <= 1.4633621;
[H5]   X5 = 1.5625000;
[H6]   X6 <= 1.4633621;
END

```

The solution, when we minimize X_3 , is:

Objective value:	.3053571400	
Variable	Value	Reduced Cost
X3	.30535714	0.0000000
X1	1.5625000	0.0000000
X2	1.5625000	0.0000000
X4	1.4633621	0.0000000
X5	1.5625000	0.0000000
X6	1.4633621	0.0000000

Thus, X_3 need not be as high as 1.4633621 in the final solution. We do find, however, that X_4 and X_6 can be no smaller than 1.4633621.

So, the final problem we want to solve is:

```

MODEL:
MIN = Z;
! The physical constraints on the X's;
[A] X1 + 2*X2 + 4*X3 + 7*X4 >= 16;
[B] 2.5*X1 + 3.5*X2 + 5.2*X5 >= 17.5;
[C] 0.4*X2 + 1.3*X4 + 7.2*X6 >= 12;
[D] 2.5*X2 + 3.5*X3 + 5.2*X5 >= 13.1;
[E] 3.5*X1 + 3.5*X4 + 5.2*X6 >= 18.2;
! Constraints for finding the minmax hurt, Z;
[H1]      X1 = 1.5625000;
[H2]      X2 = 1.5625000;
[H3] - Z + X3 = 0;
[H4]      X4 = 1.4633621;
[H5]      X5 = 1.5625000;
[H6]      + X6 = 1.4633621;
END

```

We already know the solution will be:

Variable	Value	Reduced Cost
Z	.30535714	0.0000000
X1	1.5625000	0.0000000
X2	1.5625000	0.0000000
X3	.30535714	0.0000000
X4	1.4633621	0.0000000
X5	1.5625000	0.0000000
X6	1.4633621	0.0000000

The above solution minimizes the maximum X value, as well as the number of X 's at that value. Given that maximum value (of 1.5625), it minimizes the second highest X value, as well as the number at that value; etc.

The approach described requires us to solve a sequence of linear programs. It would be nice if we could formulate a single mathematical program for finding the unordered Lexico-min. There are a number of such formulations. Unfortunately, all of them suffer from numerical problems when implemented on real computers. The formulations assume arithmetic is done with infinite precision; whereas, most computers do arithmetic with at most 15 decimal digits of precision.

14.5 Identifying Points on the Efficient Frontier

Until now, we have considered the problem of how to generate a solution on the efficient frontier. Now, let us take a slightly different perspective and consider the problem: Given a finite set of points, determine which ones are on the efficient frontier. When there are multiple criteria, it is usually impossible to find a single scoring formula to unambiguously rank all the points or players. The following table comparing on-time performance of two airlines (see Barnett, 1994) illustrates some of the issues:

Destination	Alaska Airlines		America West Airlines	
	% Arrivals	No. of Arrivals	% Arrivals	No. of Arrivals
Los Angeles	88.9	559	85.6	811
Phoenix	94.8	233	92.1	5,255
San Diego	91.4	232	85.5	448
San Francisco	83.1	605	71.3	449
Seattle	85.8	2,146	76.7	262
Weighted 5-Airport Average	86.7	3,775	89.1	7,225

The weighted average at the bottom is computed by applying a weight to the performance at airport i proportional to the number of arrivals at that airport. For example, $86.7 = (88.9 \times 559 + \dots + 85.8 \times 2146)/(559 + \dots + 2146)$.

According to this scoring, America West has a better on-time performance than Alaska Airlines. A traveler considering flying into San Francisco, however, would almost certainly prefer Alaska Airlines to America West with respect to on-time performance. In fact, the same argument applies to all five airports. Alaska Airlines dominates America West. How could America West have scored higher? The reason was a different scoring formula was used for each. Also, the airport receiving the most weight in America West's formula, sunny Phoenix, had a better on-time performance by America West than Alaska Airline's performance at its busiest airport, rainy Seattle. One should, in general, be suspicious when different scoring formulae are used for different candidates. This paradox, whereby one conclusion is supported if we look at individual groups, but the opposite conclusion is supported if we aggregate all the groups into one big group, is sometimes called Simpson's Paradox. See for example, Wagner(1982).

14.5.1 Efficient Points, More-is-Better Case

The previous example was a case of multiple performance dimensions where, for each dimension, the higher the performance number, the better the performance. We will now illustrate a method for computing a single score or number, between 0 and 1, for each player. The interpretation of this number, or efficiency score, will be that a score of 1.0 means the player or organization being measured is on the efficient frontier. In particular, there is no other player better on all dimensions or even a weighted combination of players, so the weighted averages of their performances surpass the given player on every dimension. On the other hand, a score less than 1.0 means either there is some other player better on all dimensions or there is a weighted combination of players having a weighted average performance better on all dimensions.

Define:

r_{ij} = the performance (or reward) of player i on the j^{th} dimension (e.g., the on-time performance of Alaska Airlines in Seattle);
 v_j = the weight or value to be applied to the j^{th} dimension in evaluating overall efficiency.

To evaluate the performance of player k , we will do the following in words:

Choose the v_j so as to maximize score (k)

subject to

For each player i (including k):

$$\text{score } (i) \leq 1.$$

More precisely, we want to:

$$\text{Max } \sum_j v_j r_{kj}$$

subject to

For every player i , including k :

$$\sum_j v_j r_{ij} \leq 1$$

For every weight j :

$$v_j \geq e,$$

where e is a small positive number.

The reason for requiring every v_j to be slightly positive is as follows. Suppose player k and some other player t are tied for best on one dimension, say j , but player k is worse than t on all other dimensions. Player k would like to place all the weight on dimension j , so player k will appear to be just as efficient as player t . Requiring a small positive weight on every dimension will reveal these slightly dominated players. Some care should be taken in the choice of the small “infinitesimal” constant e . If it is chosen too large, it may cause the problem to be infeasible. If it is chosen too small, it may be effectively disregarded by the optimization algorithm. From the above, you can observe that it should be bounded by:

$$e \leq 1/\sum_j r_{ij}.$$

See Mehrabian, Jahanshahloo, Alirezaee, and Amin(2000) for a more detailed discussion.

Example

The performance of five high schools in the “three R’s” of “Reading, Writing and Arithmetic” are tabulated below (see *Chicago Magazine*, February 1995):

School	Reading	Writing	Mathematics
Barrington	296	27	306
Lisle	286	27.1	322
Palatine	290	28.5	303
Hersey	298	27.3	312
Oak Park River Forest (OPRF)	294	28.1	301

Hersey, Palatine, and Lisle are clearly on the efficient frontier because they have the highest scores in reading, writing, and mathematics, respectively. Barrington is clearly not on the efficient frontier, because it is dominated by Hersey. What can we say about OPRF?

We formulate OPRF's problem as follows. Notice we have scaled both the reading and math scores, so all scores are less than 100. This is important if one requires the weight for each attribute to be at least some minimum positive value.

```

MODEL:
  MAX = 29.4*VR + 28.1*VW + 30.1*VM;
  [BAR] 29.6*VR + 27 *VW + 30.6*VM <= 1;
  [LIS] 28.6*VR + 27.1*VW + 32.2*VM <= 1;
  [PAL] 29 *VR + 28.5*VW + 30.3*VM <= 1;
  [HER] 29.8*VR + 27.3*VW + 31.2*VM <= 1;
  [OPR] 29.4*VR + 28.1*VW + 30.1*VM <= 1;
  [READ]           VR          >= 0.0005;
  [WRIT]           VW          >= 0.0005;
  [MATH]           VM          >= 0.0005;
END
```

When solved:

Optimal solution found at step: 2		
Objective value: 1.000000		
Variable	Value	Reduced Cost
VR	0.1725174E-01	0.0000000
VW	0.1700174E-01	0.0000000
VM	0.5000000E-03	0.0000000
Row	Slack or Surplus	Dual Price
1	1.000000	1.000000
BAR	0.1500157E-01	0.0000000
LIS	0.2975313E-01	0.0000000
PAL	0.0000000	0.0000000
HER	0.6150696E-02	0.0000000
OPR	0.0000000	1.000000
READ	0.1675174E-01	0.0000000
WRIT	0.1650174E-01	0.0000000
MATH	0.0000000	0.0000000

The value is 1.0, and thus, OPRF is on the efficient frontier. It should be no surprise OPRF puts the minimum possible weight on the mathematics score (where it is the lowest of the five).

14.5.2 Efficient Points, Less-is-Better Case

Some measures of performance, such as cost, are of the “less-is-better” nature. Again, we would like to have a measure of performance that gives a score of 1.0 for a player on the efficient frontier, less than 1.0 for one that is not.

Define:

c_{ij} = performance of player i on dimension j ;
 w_j = weight to be applied to the j^{th} dimension.

To evaluate the performance of player k , we want to solve a problem of the following form:

Choose weights w_j , so as to maximize the minimum weighted score,
subject to

the weighted score of player $k = 1$.

If the objective function value from this problem is less than 1, then player k is inefficient, because there is no set of weights such that player k has the best score. More precisely, we want to solve:

Max z

subject to

$$\sum_j w_j c_{kj} = 1$$

For each player i , including k :

$$\sum_j w_j c_{ij} \geq z.$$

For every weight j :

$$w_j \geq e.$$

Example

The GBS Construction Materials Company provides steel structural materials to industrial contractors. GBS recently did a survey of price, delivery performance, and quality in order to get an assessment of how it compares with its four major competitors. The results of the survey, with the names of all companies disguised, appears in the following table:

Company	Quality (based on freedom from scale, straightness, etc., based on mean rank, where 1.0 is best)	Delivery time (days)	Price (in \$/cwt)
A	1.8	14	\$21
B	4.1	1	\$26
C	3.2	3	\$25
D	1.2	5	\$23
E	2.4	7	\$22

For each of the three criteria, smaller is always better. Vendors A , B , and D are clearly competitive, based on price, delivery time, and quality, respectively. For example, a customer for whom quality is paramount will choose D . A customer for whom delivery time is important will choose B . Are C and E competitive? Imagine a customer who uses a linear weighting system to identify the best bid (e.g., score = $WQ \times \text{Quality} + WT \times (\text{delivery time}) + WP \times \text{price}$). Is there a set of weights (all nonnegative), so Score (C) < Score (i), for $i = A, B, D, E$? Likewise, for E ?

The model for Company C is:

```

MODEL:
MAX = Z;
[A] - Z + 1.8*WQ + 14*WT + 21*WP ≥ 0;
[B] - Z + 4.1*WQ +      WT + 26*WP ≥ 0;
[C] - Z + 3.2*WQ + 3*WT + 25*WP ≥ 0;
[D] - Z + 1.2*WQ + 5*WT + 23*WP ≥ 0;
[E] - Z + 2.4*WQ + 7*WT + 22*WP ≥ 0;
[CTARG] 3.2*WQ + 3*WT + 25*WP = 1;
[QUAL]   WQ ≥ 0.0005;
[TIME]   WT ≥ 0.0005;
[PRICE]  WP ≥ 0.0005;
END

```

The solution is:

Optimal solution found at step: 4		
Objective value: 0.9814257		
Variable	Value	Reduced Cost
Z	0.9814257	0.0000000
WQ	0.5000000E-03	0.0000000
WT	0.2781147E-01	0.0000000
WP	0.3659862E-01	0.0000000
Row	Slack or Surplus	Dual Price
1	0.9814257	1.000000
A	0.1774060	0.0000000
B	0.0000000	-0.5137615
C	0.1857431E-01	0.0000000
D	0.0000000	-0.4862385
E	0.1962431E-01	0.0000000
CTARG	0.0000000	0.9816514
QUAL	0.0000000	-0.4513761
TIME	0.2731147E-01	0.0000000
PRICE	0.3609862E-01	0.0000000

Company C has an efficiency rating of 0.981. Thus, it is not on the efficient frontier. With a similar model, you can show Company E is on the efficient frontier.

14.5.3 Efficient Points, the Mixed Case

In many situations, there may be some dimensions where less is better, such as risk; whereas, there are other dimensions where more is better, such as chocolate.

In this case, unless we make additional restrictions on the weights, we cannot get a simple score of efficiency between 0 and 1 for a company. We can nevertheless extend the previous approach to determine if a point is on the efficient frontier.

Define:

c_{ij} = level of the j^{th} “less is better” attribute for player i , e.g., a cost,

r_{ij} = level of the j^{th} “more is better” attribute for player i , e.g., a revenue or reward,

w_j = weight to be applied to the j^{th} “less is better” attribute,

v_j = weight to be applied to the j^{th} “more is better” attribute.

In words, to evaluate the efficiency of player or point k , we want to:

Max score (k) – (best score of any other player)
subject to
sum of the weights = 1

If the objective value is nonnegative, then player k is efficient; whereas, if the objective is negative, then there is no set of weights such that player k scores at least as well as every other player.

If we denote the best score of any other player by z , then, more specifically, we want to solve:

Max $\sum_j v_j r_{kj} - \sum_j w_j c_{kj} - z$
subject to

For each player $i, i \neq k$

$$z \geq \sum_j v_j r_{ij} - \sum_j w_j c_{kj}$$

and

$$\sum_j v_j + \sum_j w_j = 1,$$

$v_j \geq e, w_j \geq e, z$ unconstrained in sign, where e is a small positive number as introduced in the “more-is-better” case.

The dual of this problem is to find a set of nonnegative weights, λ_i , to apply to each of the other players to:

Minimize g

subject to

$$\sum_i \lambda_i = 1$$

For each “more is better” attribute j :

$$g + \sum_{i \neq k} \lambda_j r_{ij} \geq r_{kj},$$

For each “less is better” attribute j :

$$g - \sum_{i \neq k} \lambda_j c_{ij} \geq -c_{kj},$$

g unconstrained in sign.

If g is nonnegative, it means no weighted combination of other points (or players) could be found, so their weighted performance surpasses k on every dimension.

14.6 Comparing Performance with Data Envelopment Analysis

Data Envelopment Analysis (DEA) is a method for identifying efficient points in the mixed case. That is, when there are both “less is better” and “more is better” measures. An attractive feature of DEA, relative to the previous method discussed, is it does produce an efficiency score between 0 and 1. It does this by making slightly stronger assumptions about how efficiency is measured. Specifically, DEA assumes each performance measure can be classified as either an input or an output. For outputs, more is better; whereas, for inputs, less is better. The “score” of a point or a decision-making unit is then the ratio of an output score divided by an input score.

DEA was originated by Charnes, Cooper, and Rhodes (1978) as a means of evaluating the performance of decision-making units. Examples of decision-making units might be hospitals, banks, airports, schools, and managers. For example, Bessent, Bessent, Kennington, and Reagan (1982) used the approach to evaluate the performance of 167 schools around Houston, Texas. Simple comparisons can be misleading because different units are probably operating in different environments. For example, a school operating in a wealthy neighborhood will probably have higher test scores than a

school in a poor neighborhood, even though the teachers in the poor school are working harder and require more skill than the teachers in the wealthy school. Also, different decision makers may have different skills. If the teachers in school (A) are well trained in science and those in school (B) are well trained in fine arts, then a scoring system that applies a lot of weight to science may make the teachers in (B) appear to be inferior, even though they are doing an outstanding job at what they do best.

DEA circumvents both difficulties in a clever fashion. If the arts teachers were choosing the performance measures, they would choose one that placed a lot of weight on arts. However, the science teachers would probably choose a different one. DEA follows the philosophy of a popular fast food chain, that is, “Have it your way.” DEA will derive an “efficiency” score between 0 and 1 for each unit by solving the following problem:

For each unit k :

Choose a scoring function

so as to:

maximize score of unit k

subject to:

For every unit j (including k):

$\text{score}_j \leq 1$.

Thus, unit k may choose a scoring function making it look as good as possible, so long as no other unit gets a score greater than 1 when that same scoring function is applied to the other unit. If a unit k gets a score of 1.0, it means there is no other unit strictly dominating k .

In the version of DEA we consider, the allowed scoring functions are limited to ratios of weighted outputs to weighted inputs. For example:

$$\text{score} = \frac{\text{weighted sum of outputs}}{\text{weighted sum of inputs}}$$

We can normalize weights, so:

$$\text{weighted sum of inputs} = 1;$$

then “ $\text{score} \leq 1$ ” is equivalent to:

$$\text{weighted sum of outputs} \leq \text{weighted sum of inputs}.$$

Algebraically, the DEA model is:

Given

n = decision-making units,

m = number of inputs,

s = number of outputs.

Observed data:

c_{ij} = level of j^{th} input for unit i ,

r_{ij} = level of j^{th} output for unit i .

Variables:

w_j = weight applied to the j^{th} input,

v_j = weight (or value) applied to the j^{th} output.

For unit k , the model to compute the best score is:

$$\text{Maximize } \sum_{j=1}^s v_j r_{kj}$$

subject to

$$\sum_{j=1}^m w_j c_{kj} = 1$$

For each unit i (including k):

$$\sum_{j=1}^s v_j r_{ij} \leq \sum_{j=1}^m w_j c_{ij}$$

This model will tend to have more constraints than decision variables. Thus, if implementation efficiency is a major concern, one may wish to solve the dual of this model rather than the primal.

Sexton et al. (1994) describes the use of DEA to analyze the transportation efficiency of 100 county level school districts in North Carolina. Examples of inputs were number of buses used and expenses. The single output was the number of pupils transported per day. Various adjustments were made in the analysis to take into account the type of district (e.g., population density). A savings of about \$50 million over a four-year period was claimed.

Sherman and Ladino (1995) describe the use of DEA to analyze and improve the efficiency of branches in a 33-unit branch banking system. They claimed annual savings of \$6 million. Examples of inputs for a branch unit were: number of tellers, office square feet, and expenses excluding personnel. Examples of outputs were number of deposits, withdrawals, checks cashed, loans made, and new accounts. Of the 33 units, ten obtained an efficiency score of 100%. An automatic result of the DEA analysis for an inefficient unit is an identification of the one or two units that dominate the inefficient unit. This dominating unit was then used as a “benchmark or best practices case” to help identify how the inefficient unit could be improved.

Example

Below are four performance measures on six high schools: Bloom (*BL*), Homewood (*HW*), New Trier (*NT*), Oak Park (*OP*), York (*YK*), and Elgin (*EL*). Cost/pupil is the number of dollars spent per year per pupil by the school. Percent not-low-income is the fraction of the student body coming from homes not classified as low income. The writing and science scores are the averages over students in a school on a standard writing test and a standard science test. The first two measures are treated as inputs, over which teachers and administrators have no control. The test scores are treated as outputs.

School	Cost/pupil	Percent not low income	Writing score	Science score
BL	8939	64.3	25.2	223
HW	8625	99	28.2	287
NT	10813	99.6	29.4	317
OP	10638	96	26.4	291
YK	6240	96.2	27.2	295
EL	4719	79.9	25.5	222

Which schools would you consider “efficient”? New Trier has the highest score in both writing (29.4) and science (317). However, it also spends the most per pupil, \$10,813, and has the highest fraction not-low-income. A DEA model for maximizing the score of New Trier appears below. Notice we have scaled each factor, so it lies in the range (1,1000). This is important if one requires a strictly positive minimum weight on each factor, as the last four constraints of the model imply. The motivation for the strictly positive weight on each factor was given in the description of the “more-is-better” case:

```

MODEL:
MAX = SCORENT;
! Define the numerator for New Trier;
[DEFNUMNT] SCORENT - 317*WNTSCIN - 29.4*WNTWRIT = 0;
! Fix the denominator for New Trier;
[FIXDNMNT] 99.6*WNTRICH + 108.13*WNTCOST = 1;
! Numerator/ Denominator < 1 for every school,;
! or equivalently, Numerator < Denominator;
[BLNT]223*WNTSCIN+25.2*WNTWRIT-64.3*WNTRICH-89.39*WNTCOST<=0;
[HWNT]287*WNTSCIN+28.2*WNTWRIT-99*WNTRICH-86.25*WNTCOST <= 0;
[NTNT]317*WNTSCIN+29.4*WNTWRIT-99.6*WNTRICH-108.13*WNTCOST<=0;
[OPNT]291*WNTSCIN+26.4*WNTWRIT-96*WNTRICH-106.38*WNTCOST<=0;
[YKNT]295*WNTSCIN+27.2*WNTWRIT-96.2*WNTRICH-62.40*WNTCOST<=0;
[ELNT]222*WNTSCIN+25.5*WNTWRIT-79.9*WNTRICH-47.19*WNTCOST<=0;
! Each measure must receive a little weight;
[SCINT] WNTSCIN >= 0.0005;
[WRINT] WNTWRIT >= 0.0005;
[RICNT] WNTRICH >= 0.0005;
[COSNT] WNTCOST >= 0.0005;
END

```

The solution is:

Optimal solution found at step:	3	
Objective value:	0.9615803	
Variable	Value	Reduced Cost
SCORENT	0.9615803	0.0000000
WNTSCIN	0.2987004E-02	0.0000000
WNTWRIT	0.5000000E-03	0.0000000
WNTRICH	0.8204092E-02	0.0000000
WNTCOST	0.1691228E-02	0.0000000
Row	Slack or Surplus	Dual Price
1	0.9615803	1.0000000
DEFNUMNT	0.0000000	1.0000000
FIXDNMNT	0.0000000	0.9635345
BLNT	0.0000000	0.8795257
HWNT	0.8670327E-01	0.0000000
NTNT	0.3841965E-01	0.0000000
OPNT	0.8508738E-01	0.0000000
YKNT	0.0000000	0.4097145
ELNT	0.5945104E-01	0.0000000
SCINT	0.2487004E-02	0.0000000
WRINT	0.0000000	-3.908281
RICNT	0.7704092E-02	0.0000000
COSNT	0.1191227E-02	0.0000000

The score of New Trier is less than 1.0. Thus, according to DEA, New Trier is not efficient. Looking at the solution report, one can deduce that *NT* is, according to DEA, strictly less efficient than *BL* and *YK*. Notice their “score less-than-or-equal-to 1” constraints are binding. Thus, if *NT* wants to improve its efficiency by doing a benchmark study, it should perhaps study the practices of *BL* and *YK* for insight.

A sets-based model that evaluates all the schools in one model is given below:

```

MODEL:
! Data Envelopment Analysis of Decision Maker Efficiency ;
SETS:
  DMU:      !The decisionmaking units;
  SCORE; ! Each decision making unit has a
          score to be computed;
  FACTOR;
! There is a set of factors, input & output;
  DXF( DMU, FACTOR): F, ! F( I, J) = Jth factor of DMU I;
                      W; ! Weights used to compute DMU I's score;
ENDSETS
DATA:
  DMU = BL HW NT OP YK EL;
! Inputs are spending/pupil, % not low income;
! Outputs are Writing score and Science score;
  NINPUTS = 2; ! The first NINPUTS factors are inputs;
  FACTOR= COST RICH      WRIT SCIN;
! The inputs,    the outputs;
  F   = 89.39  64.3    25.2   223
        86.25  99     28.2   287
        108.13 99.6    29.4   317
        106.38 96     26.4   291
        62.40  96.2    27.2   295
        47.19  79.9    25.5   222;
  WGTMIN = .0005; ! Min weight applied to every factor;
  BIGM = 999999; ! Biggest a weight can be;
ENDDATA
!-----
! The Model;
! Try to make everyone's score as high as possible;
  MAX = @SUM( DMU: SCORE);
! The LP for each DMU to get its score;
  @FOR( DMU( I):
    [CSCR] SCORE( I) = @SUM( FACTOR(J)|J #GT# NINPUTS:
                           F(I, J)* W(I, J));
  ! Sum of inputs(denominator) = 1;
    [SUM21] @SUM( FACTOR( J)| J #LE# NINPUTS:
                  F( I, J)* W( I, J)) = 1;
  ! Using DMU I's weights, no DMU can score better than 1,
  Note Numer/Denom <= 1 implies Numer <= Denom;
  @FOR( DMU( K):
    [LE1] @SUM( FACTOR( J)| J #GT# NINPUTS: F( K, J) * W( I, J))
          <= @SUM( FACTOR( J)| J #LE# NINPUTS: F( K, J) * W( I, J))
    )
  );
! The weights must be greater than zero;

```

```

@FOR( DXF( I, J): @BND( WGTMIN, W, BIGM));
END

```

Part of the output is:

Variable	Value	Reduced Cost
SCORE(BL)	1.000000	0.0000000
SCORE(HW)	0.9095071	0.0000000
SCORE(NT)	0.9615803	0.0000000
SCORE(OP)	0.9121280	0.0000000
SCORE(YK)	1.000000	0.0000000
SCORE(EL)	1.000000	0.0000000

We see that the only efficient schools are Bloom, Yorktown, and Elgin.

14.7 Problems

- In the example staffing problem in this chapter, the primary criterion was minimizing the number of people hired. The secondary criterion was to spread out any excess capacity as much as possible. The primary criterion received a weight of 9; whereas, the secondary criterion received a weight of 1. The minimum number of people required (primary criterion) was 8. How much could the weight on the secondary criterion be increased before the number of people hired increases to more than 8?
- Reconsider the advertising media selection problem of this chapter.
 - Reformulate it, so we achieve at least 197 (in 1000's) useful exposures at minimum cost.
 - Predict the cost before looking at the solution.
- A description of a “project crashing” decision appears in Chapter 8. There were two criteria, project length and project cost. Trace out the efficient frontier describing the trade-off between length and cost.
- The various capacities of several popular sport utility vehicles, as reported by a popular consumer rating magazine, are listed below:

Vehicle	Seats	Cargo Floor Length (in.)	Rear Opening Height (in.)	Cargo Volume (cubic ft)
Blazer	6	75.5	31.5	42.5
Cherokee	5	62.0	33.5	34.5
Land Rover	7	49.5	42.0	42.0
Land Cruiser	8	65.5	38.5	44.5
Explorer	6	78.5	35.0	48.0
Trooper	5	57.0	36.5	42.5

Assuming sport utility vehicle buyers sport a linear utility and more capacity is better, which of the above vehicles are on the efficient frontier according to these four capacity measures?

5. The Rotorua Fruit Company sells various kinds of premium fruits (e.g., apples, peaches, and kiwi fruit) in small boxes. Each box contains a single kind of fruit. The outside of the box specifies:

- i) the kind of fruit,
- ii) the number of pieces of fruit, and
- iii) the approximate weight of the fruit in the box.

Satisfying specification (iii) is nontrivial, because the per unit weight of fruit as it comes from the orchard is a random variable. Consider the case of apples. Each apple box contains 12 apples. The label on each apple box says the box contains 4.25 lbs. of apples. In fact, a typical apple weighs from 5 to 6.5 ounces. At 16 ounces/lb., a box of 5-ounce apples would weigh only 3.75 lbs., whereas, a box of 6.5-ounce apples would weigh 4.875 lbs. The approach Rotorua is considering is to have a set of 24 automated scales on the box loading line. The 24 scales will be loaded with 24 apples. Based on the weights of the apples, a set of 12 apples whose total weight comes close to 4.25 lbs., will be dropped into the current empty box. In the next cycle, the 12 empty scales will be reloaded with new apples, a new empty box will be moved into position, and the process repeated. Rotorua cannot always achieve the ideal of exactly 4.25 lbs. in a box. However, being underweight is worse than being overweight. Rotorua has characterized its feeling/utility for this under/over issue by stating that given the choice between loading a box one ounce under and one ounce over, it clearly prefers to load it one ounce over. However, it would be indifferent between loading a box one ounce under vs. five ounces over.

Suppose the scales currently contain apples with the following weights in ounces:

5.6, 5.9, 6.0, 5.8, 5.9, 5.4, 5.0, 5.5, 6.3, 6.2, 5.1, 6.2,
6.1, 5.2, 6.4, 5.7, 5.6, 5.5, 5.3, 6.0, 5.4, 5.3, 5.8, 6.1.

- a) How would you load the next box?
- b) Discuss some of the issues in implementing your approach.

16

Game Theory and Cost Allocation

16.1 Introduction

In most decision-making situations, our profits (and losses) are determined not only by our decisions, but by the decisions taken by outside forces (e.g., our competitors, the weather, etc.). A useful classification is whether the outside force is indifferent or mischievous. We, for example, classify the weather as indifferent because its decision is indifferent to our actions, in spite of how we might feel during a rainstorm after washing the car and forgetting the umbrella. A competitor, however, generally takes into account the likelihood of our taking various decisions and as a result tends to make decisions that are mischievous relative to our welfare. In this chapter, we analyze situations involving a mischievous outside force. The standard terminology applied to the problem to be considered is *game theory*. Situations in which these problems might arise are in the choice of a marketing or price strategy, international affairs, military combat, and many negotiation situations. For example, the probability a competitor executes an oil embargo against us probably depends upon whether we have elected a strategy of building up a strategic petroleum reserve. Frequently, the essential part of the problem is deciding how two or more cooperating parties “split the pie”. That is, allocate costs or profits of a joint project. For a thorough introduction to game theory, see Fudenberg and Tirole (1993).

16.2 Two-Person Games

In so-called two-person game theory, the key feature is *each of the two players must make a crucial decision ignorant of the other player's decision*. Only after both players have committed to their respective decisions does each player learn of the other player's decision and each player receives a payoff that depends solely on the two decisions. Two-person game theory is further classified according to whether the payoffs are constant sum or variable sum. In a constant sum game, the total payoff summed over both players is constant. Usually this constant is assumed to be zero, so one player's gain is exactly the other player's loss. The following example illustrates a constant sum game.

A game is to be played between two players called *Blue* and *Gold*. It is a single simultaneous move game. Each player must make her single move in ignorance of the other player's move. Both moves are then revealed and then one player pays the other an amount specified by the payoff table below:

		Payoff from Blue to Gold	
		Blue's Move	
		a	b
Gold's Move	a	4	-6
	b	-5	8
	c	3	-4

Blue must choose one of two moves, (a) or (b), while Gold has a choice among three moves, (a), (b), or (c). For example, if Gold chooses move (b) and Blue chooses move (a), then Gold pays Blue 5 million dollars. If Gold chooses (c) and Blue chooses (a), then Blue pays Gold 3 million dollars.

16.2.1 The Minimax Strategy

This game does not have an obvious strategy for either player. If Gold is tempted to make move (b) in the hopes of winning the 8 million dollar prize, then Blue will be equally tempted to make move (a), so as to win 5 million from Gold. For this example, it is clear each player will want to consider a random strategy. Any player who follows a pure strategy of always making the same move is easily beaten. Therefore, define:

$$BM_i = \text{probability Blue makes move } i, i = a \text{ or } b,$$

$$GM_i = \text{probability Gold makes move } i, i = a, b, \text{ or } c.$$

How should Blue choose the probabilities BM_i ? Blue might observe that:

If Gold chooses move (a), my expected loss is:

$$4 BMA - 6 BMB.$$

If Gold chooses move (b), my expected loss is:

$$-5 BMA + 8 BMB.$$

If Gold chooses move (c), my expected loss is:

$$3 BMA - 4 BMB.$$

So, there are three possible expected losses depending upon which decision is made by Gold. If Blue is conservative, a reasonable criterion is to choose the BM_i , so as to minimize the maximum expected loss. This philosophy is called the *minimax strategy*. Stated another way, Blue wants to choose the probabilities BM_i , so, no matter what Gold does, Blue's maximum expected loss is minimized. If LB is the maximum expected loss to Blue, the problem can be stated as the LP:

```

MIN = LB;
! Probabilities must sum to 1;
    BMA      + BMB  = 1;
! Expected loss if Gold chooses (a);
    -LB + 4 * BMA - 6 * BMB <= 0;
! Expected loss if Gold chooses (b);
    -LB - 5 * BMA + 8 * BMB <= 0;
! Expected loss if Gold chooses (c);
    -LB + 3 * BMA - 4 * BMB <= 0;

```

The solution is:

Optimal solution found at step: 2		
Objective value:		0.2000000
Variable	Value	Reduced Cost
LB	0.2000000	0.0000000
BMA	0.6000000	0.0000000
BMB	0.4000000	0.0000000
Row	Slack or Surplus	Dual Price
1	0.2000000	1.0000000
2	0.0000000	-0.2000000
3	0.2000000	0.0000000
4	0.0000000	0.3500000
5	0.0000000	0.6500000

The interpretation is, if Blue chooses move (a) with probability 0.6 and move (b) with probability 0.4, then Blue's expected loss is never greater than 0.2, regardless of Gold's move.

If Gold follows a similar argument, but phrases the argument in terms of maximizing the minimum expected profit, PG , instead of minimizing maximum loss, then Gold's problem is:

```
MAX = PG;
! Probabilities sum to 1;
GMA + GMB + GMC = 1;
! Expected profit if Blue chooses (a);
-PG + 4 * GMA - 5 * GMB + 3 * GMC >= 0;
! Expected profit if Blue chooses (b);
-PG - 6 * GMA + 8 * GMB - 4 * GMC >= 0;
```

The solution to Gold's problem is:

Optimal solution found at step: 1		
Objective value:		0.2000000
Variable	Value	Reduced Cost
PG	0.2000000	0.0000000
GMA	0.0000000	0.1999999
GMB	0.3500000	0.0000000
GMC	0.6500000	0.0000000
Row	Slack or Surplus	Dual Price
1	0.2000000	1.0000000
2	0.0000000	0.2000000
3	0.0000000	-0.6000000
4	0.0000000	-0.4000000

The interpretation is, if Gold chooses move (b) with probability 0.35, move (c) with probability 0.65 and never move (a), then Gold's expected profit is never less than 0.2. Notice Gold's lowest expected profit equals Blue's highest expected loss. From Blue's point of view the expected transfer to Gold is at least 0.2. The only possible expected transfer is then 0.2. This means if both players follow the random strategies just derived, then on every play of the game there is an expected transfer of 0.2 units from Blue to Gold. The game is biased in Gold's favor at the rate of 0.2 million dollars per play. The strategy of randomly choosing among alternatives to keep the opponent guessing, is sometimes also known as a mixed strategy.

If you look closely at the solutions to Blue's LP and to Gold's LP, you will note a surprising similarity. The dual prices of Blue's LP equal the probabilities in Gold's LP and the negatives of

Gold's dual prices equal the probabilities of Blue's LP. Looking more closely, you can note each LP is really the dual of the other one. This is always true for a two-person game of the type just considered and mathematicians have long been excited by this fact.

16.3 Two-Person Non-Constant Sum Games

There are many situations where the welfare, utility, or profit of one person depends not only on his decisions, but also on the decisions of others. A two-person game is a special case of the above in which:

1. there are exactly two players/decision makers,
2. each must make one decision,
3. in ignorance of the other's decision, and
4. the loss incurred by each is a function of both decisions.

A two-person constant sum game (frequently more narrowly called a zero sum game) is the special case of the above where:

- (4a) the losses to both are in the same commodity (e.g., dollars) and
- (4b) the total loss is a constant independent of players' decisions.

Thus, in a constant sum game the sole effect of the decisions is to determine how a "constant sized pie" is allocated. Ordinary linear programming can be used to solve two-person constant sum games.

When (1), (2) and (3) apply, but (4b) does not, then we have a two-person non-constant sum game. Ordinary linear programming cannot be used to solve these games. However, closely related algorithms, known as linear complementarity algorithms, are commonly applied. Sometimes a two-person non-constant sum game is also called a bimatrix game.

As an example, consider two firms, each of which is about to introduce an improved version of an already popular consumer product. The versions are very similar, so one firm's profit is very much affected by its own advertising decision as well as the decision of its competitor. The major decision for each firm is presumed to be simply the level of advertising. Suppose the losses (in millions of dollars) as a function of decision are given by Figure 16.1. The example illustrates that each player need not have exactly the same kinds of alternatives.

Figure 16.1 Two Person, Non-constant Sum Game

		Firm A		
		No Advertise	Advertise Medium	Advertise High
		No Advertise	-4	-3
Firm B	No Advertise	-4	-2	1
	Advertise	-1	-2	-1
		No Advertise	-5	-1
		Advertise	0	

Negative losses correspond to profits.

16.3.1 Prisoner's Dilemma

This cost matrix has the so-called prisoner's dilemma cost structure. This name arises from a setting in which two accomplices in crime find themselves in separate jail cells. If neither prisoner cooperates with the authorities (thus the two cooperate), both will receive a medium punishment. If one of them provides evidence against the other, the other will get severe punishment while the one who provides evidence will get light punishment, *if the other does not provide evidence against the first*. If each provides evidence against the other, they both receive severe punishment. Clearly, the best thing for the two as a group is for the two to cooperate with each other. However, individually there is a strong temptation to defect.

The prisoner's dilemma is common in practice, especially in advertising. The only way of getting to Mackinac Island in northern Michigan is via ferry from Mackinaw City. Three different companies, Sheplers, the Arnold Line, and the Star Line operate such ferries. As you approach Mackinaw City by car, you may notice up to a mile before the ferry landing, that each company has one or more small roadside stands offering to sell ferry tickets for their line. Frequent users of the ferry service proceed directly to the well-marked dock area and buy a ticket after parking the car and just before boarding the ferry (no cars are allowed on Mackinac Island). No reserved seats are sold, so there is no advantage to buying the tickets in advance at the stands. First time visitors, however, are tempted to buy a ticket at a company specific stand because the signs suggest that this is the safe thing to do. The "socially" most efficient arrangement would be to have no advanced ticket booths. If a company does not have a stand, however, while its competitors do, then this company will lose a significant fraction of the first time visitor market.

The same situation exists with the two firms in our little numerical example. For example, if *A* does not advertise, but *B* does, then *A* makes 1 million and *B* makes 5 million of profit. Total profit

would be maximized if neither advertised. However, if either knew the other would not advertise, then the one who thought he had such clairvoyance would have a temptation to advertise.

Later, it will be useful to have a loss table with all entries strictly positive. The relative attractiveness of an alternative is not affected if the same constant is added to all entries. Figure 16.2 was obtained by adding +6 to every entry in Figure 16.1:

Figure 16.2 Two Person, Non-constant Sum Game

		Firm A		
		No Advertise	Advertise Medium	Advertise High
Firm B	No Advertise	2	3	1
	Advertise	2	4	7
	No Advertise	5	4	5
	Advertise	1	5	6

We will henceforth work with the data in Figure 16.2.

16.3.2 Choosing a Strategy

Our example illustrates that we might wish our own choice to be:

- i. somewhat unpredictable by our competitor, and
- ii. robust in the sense that, regardless of how unpredictable our competitor is, our expected profit is high.

Thus, we are lead (again) to the idea of a random or mixed strategy. By making our decision random (e.g., by flipping a coin) we tend to satisfy (i). By biasing the coin appropriately, we tend to satisfy (ii).

For our example, define a_1, a_2, a_3 as the probability A chooses the alternative “No advertise”, “Advertise Medium”, and “Advertise High”, respectively. Similarly, b_1 and b_2 are the probabilities that B applies to alternatives “No Advertise” and “Advertise”, respectively. How should firm A choose a_1, a_2 , and a_3 ? How should firm B choose b_1 and b_2 ?

For a bimatrix game, it is difficult to define a solution that is simultaneously optimum for both. We can, however, define an equilibrium stable set of strategies. A stable solution has the feature that, given B 's choice for b_1 and b_2 , A is not motivated to change his probabilities a_1, a_2 , and a_3 . Likewise, given a_1, a_2 , and a_3 , B is not motivated to change b_1 and b_2 . Such a solution, where no player is

motivated to unilaterally change his or her strategy, is sometimes also known as a Nash equilibrium. There may be bimatrix games with several stable solutions.

What can we say beforehand about a strategy of A 's that is stable? Some of the a_i 's may be zero while for others we may have $a_i > 0$. An important observation which is not immediately obvious is the following: the expected loss to A of choosing alternative i is the same over all i for which $a_i > 0$. If this were not true, then A could reduce his overall expected loss by increasing the probability associated with the lower loss alternative. Denote the expected loss to A by v_A . Also, the fact that $a_i = 0$ must imply the expected loss from choosing i is $> v_A$. These observations imply that, with regard to A 's behavior, we must have:

$$2b_1 + 5b_2 \geq v_A \text{ (with equality if } a_1 > 0\text{),}$$

$$3b_1 + 4b_2 \geq v_A \text{ (with equality if } a_2 > 0\text{),}$$

$$b_1 + 5b_2 \geq v_A \text{ (with equality if } a_3 > 0\text{).}$$

Symmetric arguments for B imply:

$$2a_1 + 4a_2 + 7a_3 \geq v_B \text{ (with equality if } b_1 > 0\text{),}$$

$$a_1 + 5a_2 + 6a_3 \geq v_B \text{ (with equality if } b_2 > 0\text{).}$$

We also have the nonnegativity constraints:

$$a_i \geq 0 \text{ and } b_i \geq 0, \text{ for all alternatives } i.$$

Because the a_i and b_i are probabilities, we wish to add the constraints $a_1 + a_2 + a_3 = 1$ and $b_1 + b_2 = 1$.

If we explicitly add slack (or surplus if you wish) variables, we can write:

$$2b_1 + 5b_2 - slka1 = v_A$$

$$3b_1 + 4b_2 - slka2 = v_A$$

$$b_1 + 5b_2 - slka3 = v_A$$

$$2a_1 + 4a_2 + 7a_3 - slkb1 = v_B$$

$$a_1 + 5a_2 + 6a_3 - slkb2 = v_B$$

$$a_1 + a_2 + a_3 = 1$$

$$b_1 + b_2 = 1$$

$$a_i \geq 0, b_i \geq 0, slka_i \geq 0, \text{ and } slkb_i \geq 0, \text{ for all alternatives } i.$$

$$slka1^* a1 = 0$$

$$slka2^* a2 = 0$$

$$slka3^* a3 = 0$$

$$slkb1^* b1 = 0$$

$$slkb2^* b2 = 0$$

The last five constraints are known as the complementarity conditions. The entire model is known as a linear complementarity problem.

Rather than use a specialized linear complementarity algorithm, we will simply use the integer programming capabilities for LINGO to model the problem as follows:

```

MODEL: ! Two person nonconstant sum game. (BIMATRIX);
SETS:
  OPTA: PA, SLKA, NOTUA, COSA;
  OPTB: PB, SLKB, NOTUB, COSB;
  BXA( OPTB, OPTA): C2A, C2B;
ENDSETS
DATA:
OPTB = BNAD BYAD;
OPTA = ANAD AMAD AHAD;
C2A =  2      3      1 ! C2A( I, J) = cost to A if B;
           5      4      5; ! chooses row I, A chooses col J;
C2B =  2      4      7 ! C2B( I, J) = cost to B if B;
           1      5      6; ! chooses row I, A chooses col J;
ENDDATA
!-----;
! Conditions for A, for each option J;
@FOR( OPTA( J):
! Set CBSTA= cost of strategy J, if J is used by A;
  CBSTA = COSA( J) - SLKA( J);
  COSA( J) = @SUM( OPTB( I): C2A( I, J) * PB( I));
! Force SLKA( J) = 0 if strategy J is used;
  SLKA( J) <= NOTUA( J) * @MAX( OPTB( I):
    C2A( I, J));
! NOTUA( J) = 1 if strategy J is not used;
  PA( J) <= 1 - NOTUA( J);
! Either strategy J is used or it is not used;
  @BIN( NOTUA( J));
);
! A must make a decision;
@SUM( OPTA( J): PA( J)) = 1;
! Conditions for B;
@FOR( OPTB( I):
! Set CBSTB = cost of strategy I, if I is used by
  B;
  CBSTB = COSB( I) - SLKB( I);
  COSB( I) = @SUM( OPTA( J): C2B( I, J) * PA( J));
! Force SLKB( I) = 0 if strategy I is used;
  SLKB( I) <= NOTUB( I) * @MAX( OPTA( J):
    C2B( I, J));
! NOTUB( I) = 1 if strategy I is not used;
  PB( I) <= 1 - NOTUB( I);
! Either strategy I is used or it is not used;
  @BIN( NOTUB( I));
);
! B must make a decision;
@SUM( OPTB( I): PB( I)) = 1;
END

```

A solution is:

Variable	Value
CBSTA	3.666667
CBSTB	5.500000
PA (AMAD)	0.500000
PA (AHAD)	0.500000
SLKA (ANAD)	0.3333333
NOTUA (ANAD)	1.000000
COSA (ANAD)	4.000000
COSA (AMAD)	3.666667
COSA (AHAD)	3.666667
PB (BNAD)	0.3333333
PB (BYAD)	0.6666667
COSB (BNAD)	5.500000
COSB (BYAD)	5.500000

The solution indicates that firm *A* should not use option 1(No ads) and should randomly choose with equal probability between options 2 and 3. Firm *B* should choose its option 2(Advertise) twice as frequently as it chooses its option 1(Do not advertise).

The objective function value, reduced costs and dual prices can be disregarded. Using our original loss table, we can calculate the following:

Situation		Probability	Weighted Contribution To Total Loss of	
A	B		A	B
No Ads	No Ads	$0 \times 1/3$	0	0
No Ads	Ads	$0 \times 2/3$	0	0
Advertise Medium	No Ads	$1/2 \times 1/3$	$(1/6) \times (-3)$	$(1/6) \times (-2)$
Advertise Medium	Ads	$1/2 \times 2/3$	$(1/3) \times (-2)$	$(1/3) \times (-1)$
Advertise High	No Ads	$1/2 \times 1/3$	$(1/6) \times (-5)$	$(1/6) \times (1)$
Advertise High	Ads	$1/2 \times 2/3$	$(1/3) \times (-1)$	$(1/3) \times (0)$
			-2.3333	-0.5

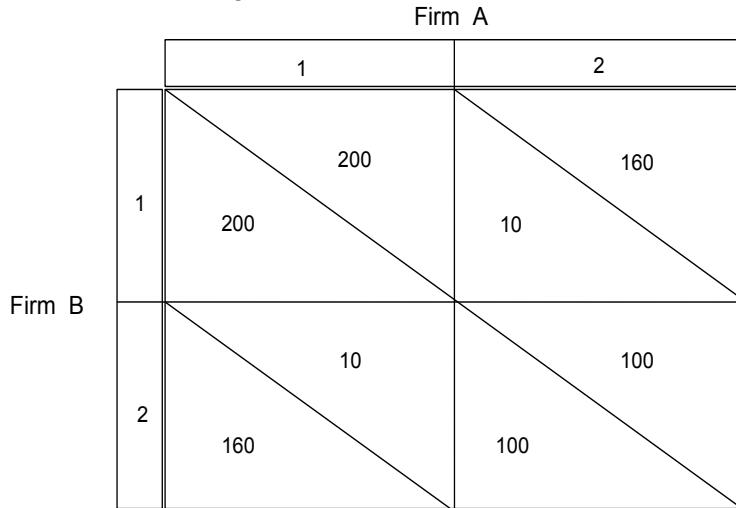
Thus, following the randomized strategy suggested, *A* would have an expected profit of 2.33 million; whereas, *B* would have an expected profit of 0.5 million. Contrast this with the fact that, if *A* and *B* cooperated, they could each have an expected profit of 4 million.

16.3.3 Bimatrix Games with Several Solutions

When a nonconstant sum game has multiple or alternative stable solutions, life gets more complicated. The essential observation is we must look outside our narrow definition of “stable solution” to decide which of the stable solutions, if any, would be selected in reality.

Consider the following nonconstant sum two-person game:

Figure 16.3 Bimatrix Games



As before, the numbers represent losses.

First, observe the one solution that is stable according to our definition: (I) Firm *A* always chooses option 1 and Firm *B* always chooses option 2. Firm *A* is not motivated to switch to 2 because its losses would increase to 100 from 10. Similarly, *B* would not switch to 1 from 2 because its losses would increase to 200 from 160. The game is symmetric in the players, so similar arguments apply to the solution (II): *B* always chooses 1 and *A* always chooses 2.

Which solution would result in reality? It probably depends upon such things as the relative wealth of the two firms. Suppose:

- i. *A* is the wealthier firm,
- ii. the game is repeated week after week, and
- iii. currently the firms are using solution *II*.

After some very elementary analysis, *A* concludes it much prefers solution *I*. To move things in this direction, *A* switches to option 1. Now, it becomes what applied mathematicians call a game of "chicken". Both players are taking punishment at the rate of 200 per week. Either player could improve its lot by $200 - 160 = 40$ by unilaterally switching to its option 2. However, its lot would be improved a lot more (i.e., $200 - 10 = 190$) if its opponent unilaterally switched. At this point, a rational *B* would probably take a glance at *A*'s balance sheet and decide *B* switching to option 2 is not such a bad decision. When a game theory problem has multiple solutions, any given player would like to choose that stable solution which is best for it. If the player has the wherewithal to force such a solution (e.g., because of its financial size), then this solution is sometimes called a Stackelberg equilibrium.

If it is not clear which firm is wealthier, then the two firms may decide a cooperative solution is best (e.g., alternate between solutions *I* and *II* in alternate weeks). At this point, however, federal antitrust authorities might express a keen interest in this bimatrix game.

We conclude a “stable” solution is stable only in a local sense. When there are multiple stable solutions, we should really look at all of them and take into account other considerations in addition to the loss matrix.

16.4 Nonconstant-Sum Games Involving Two or More Players

The most unrealistic assumption underlying classical two-person constant-sum game theory is the sum of the payoffs to all players must sum to zero (actually a constant, without loss of generality). In reality, the total benefits are almost never constant. Usually, total benefits increase if the players cooperate, so these situations are sometimes called cooperative games. In these nonconstant-sum games, the difficulty then becomes one of deciding how these additional benefits due to cooperation should be distributed among the players.

There are two styles for analyzing nonconstant sum games. If we restrict ourselves to two persons, then so-called bimatrix game theory extends the methods for two-person constant sum games to nonconstant sum games. If there are three or more players, then *n-person game theory* can be used in selecting a decision strategy. The following example illustrates the essential concepts of *n-person game theory*.

Three property owners, *A*, *B*, and *C*, own adjacent lakefront property on a large lake. A piece of property on a large lake has higher value if it is protected from wave action by a seawall. *A*, *B*, and *C* are each considering building a seawall on their properties. A seawall is cheaper to build on a given piece of property if either or both of the neighbors have seawalls. For our example, *A* and *C* already have expensive buildings on their properties. *B* does not have buildings and separates *A* from *C* (i.e., *B* is between *A* and *C*). The net benefits of a seawall for the three owners are summarized below:

Owners Who Cooperate, i.e., Build While Others Do Not	Net Benefit to Cooperating Owners
<i>A</i> alone	1.2
<i>B</i> alone	0
<i>C</i> alone	1
<i>A</i> and <i>B</i>	4
<i>A</i> and <i>C</i>	3
<i>B</i> and <i>C</i>	4
<i>A</i> , <i>B</i> , and <i>C</i>	7

Obviously, all three owners should cooperate and build a unified seawall because then their total benefits will be maximized. It appears *B* should be compensated in some manner because he has no motivation to build a seawall by himself. Linear programming can provide some help in selecting an acceptable allocation of benefits.

Denote by v_A , v_B , and v_C the net benefits, which are to be allocated to owners A , B , and C . No owner or set of owners will accept an allocation that is less than that, which they would enjoy if they acted alone. Thus, we can conclude:

$$\begin{aligned}v_A &\geq 1.2 \\v_B &\geq 0 \\v_C &\geq 1 \\v_A + v_B &\geq 4 \\v_A + v_C &\geq 3 \\v_B + v_C &\geq 4 \\v_A + v_B + v_C &\leq 7\end{aligned}$$

That is, any allocation satisfying the above constraints should be self-enforcing. No owner would be motivated to not cooperate. He cannot do better by himself. The above constraints describe what is called the “core” of the game. Any solution (e.g., $v_A = 3$, $v_B = 1$, $v_C = 3$) satisfying these constraints is said to be in the core.

Various objective functions might be appended to this set of constraints to give an LP. The objective could take into account secondary considerations. For example, we might choose to maximize the minimum benefit. The LP in this case is:

$$\begin{aligned}\text{Maximize } z \\ \text{subject to } z \leq v_A; z \leq v_B; z \leq v_C \\ v_A \geq 1.2 \\ v_C \geq 1 \\ v_A + v_B \geq 4 \\ v_A + v_C \geq 3 \\ v_A + v_B + v_C \leq 7\end{aligned}$$

A solution is $v_A = v_B = v_C = 2.3333$.

Note the core can be empty. That is, there is no feasible solution. This would be true, for example, if the value of the coalition A , B , C was 5.4 rather than 7. This situation is rather interesting. Total benefits are maximized by everyone cooperating. However, total cooperation is inherently unstable when benefits are 5.4. There will always be a pair of players who find it advantageous to form a subcoalition and improve their benefits (at the considerable expense of the player left out). As an example, suppose the allocations to A , B , and C under full cooperation are 1.2, 2.1, and 2.1, respectively. At this point, A would suggest to B the two of them exclude C and cooperate between the two of them. A would suggest to B the allocation of 1.8, 2.2, and 1. This is consistent with the fact that A and B can achieve a total of 4 when cooperating. At this point, C might suggest to A that the two of them cooperate and thereby select an allocation of 1.9, 0, 1.1. This is inconsistent with the fact that A and C can achieve a total of 3 when cooperating. At this point, B suggests to C etc. Thus, when the core is empty, it may be everyone agrees that full cooperation can be better for everyone. There nevertheless must be an enforcement mechanism to prevent “greedy” members from pulling out of the coalition.

16.4.1 Shapley Value

Another popular allocation method for cooperative games is the Shapley Value. The rule for the Shapley Value allocation is that each player should be awarded his average marginal contribution to the coalition if one considers all possible sequences for forming the full coalition. The following table illustrates for the previous example:

<u>Sequence</u>	Marginal value of player		
	A	B	C
A B C	1.2	2.8	3
A C B	1.2	4	1.8
B A C	4	0	3
B C A	3	0	4
C A B	2	4	1
C B A	<u>3</u>	<u>3</u>	<u>1</u>
Total:	14.4	13.8	13.8
Average:	2.4	2.3	2.3

Thus, the Shapley value allocates slightly more, 2.4, to Player *A* in our example. For this example, as with most typical cooperative games, the Shapley Value allocation is in the core if the core is non-empty.

16.5 The Stable Marriage/Assignment Problem

The stable marriage problem is the multi-person interpretation of the assignment problem. Although the major application of the stable marriage model is in college admissions and large scale labor markets, the problem historically has been explained as the “marriage” problem of assigning each of n men to exactly one of n women, and vice versa. Instead of there being a single objective function, each man provides a preference ranking of each of the women, and each woman provides a ranking of each of the men. An assignment is said to be stable if for every man i , and woman j , either: a) man i prefers the woman he currently is assigned to over woman j , or b) woman j prefers the man she is currently assigned to over man i . Otherwise, man i and woman j would be motivated to abandon their current partners and “elope”. The stable marriage assignment method has been used for assigning medical residents and interns to hospitals in the U.S. since 1952. Each year, thousands of prospective interns rank each of the hospitals in which they are interested, and each hospital ranks each of the interns in which they are interested. Then a neutral agency, the National Resident Matching Program (NRMP) assigns interns to hospitals using methods described below. A similar system is used in Canada and Scotland. Norway and Singapore use a similar approach to assign students to schools and universities. Roth (1984) gives a very interesting history of how the U.S. medical profession came to use the stable marriage assignment method embodied in NRMP. Roth, Sonmez, and Unver(2005) describe the establishment of a system, based on the marriage assignment method, for matching kidney donors with people needing kidneys.

In any multi-player problem, the following questions should always be asked: a) Is there always a stable assignment or more generally an equilibrium solution? b) Can there be multiple stable solutions? c) If yes, what criterion should we use for choosing among the multiple solutions? d) Is the solution Pareto optimal, i.e., undominated? e) Is our method for solving the problem, in particular how we answer (c), incentive compatible? That is, does the method motivate the players to provide accurate input information, e.g., rankings, for our method?

We illustrate ideas with the following 3-man, 3-woman example from Gale and Shapley(1962). A 1 means most attractive. A 3 means least attractive.

```

MAN = ADAM BOB CHUCK;
WOMAN= ALICE BARB CARMEN;
! Men(row) preference for women(col);
MPREF =
    1    2    3 !ADAM;
    3    1    2 !BOB;
    2    3    1; !CHUCK;
! Women(col) preference for men(row);
WPREF =
    3    2    1 !ADAM;
    1    3    2 !BOB;
    2    1    3;!CHUCK;
! Thus, Adam's first choice is Alice.
Alice's first choice is Bob;

```

We shall see from this example that the answer to question (b) is that, yes, there can be multiple stable solutions. In this example, giving each man his first choice (and incidentally, each woman her third choice) is feasible, giving the assignment: Adam with Alice, Bob with Barb, and Chuck with Carmen. It is stable because no man is motivated to switch. A second stable solution is possible by giving each woman her first choice (and incidentally, each man his third choice), namely: Adam with Carmen, Bob with Alice, and Chuck with Barb. It is stable because no woman is motivated to switch. A third, less obvious stable solution is to give everyone their second choice: Adam with Barb, Bob with Carmen, and Chuck with Alice. All other assignments are unstable.

How to solve the problem?

Gale and Shapley (1962) show that an intuitive iterative courtship type of method can be made into a rigorous algorithm for finding a stable assignment. The algorithm proceeds as follows:

- 1) Each man proposes, or is tentatively assigned, to his first choice woman.
- 2) If every woman has exactly one man assigned to her, then stop. We have a stable assignment.
- 3) Else, each woman who has two or more men assigned to her rejects all but one of the men assigned to her, tentatively keeping the one most attractive to her of the men that just proposed to her.
- 4) Each man just rejected in (3) proposes/is assigned to the next most attractive woman on his list.
- 5) Go to (2).

This version of the algorithm will produce the first solution mentioned above in which all men get their first choice. Obviously, there is the female version of this algorithm in which the roles of men and women are exchanged. That version gives the second solution above. Gale and Shapley(1962) make the following observations: i) Regarding our question (a) above, this algorithm will always find a stable solution; ii) If both the male and the female versions of the algorithm give the same assignment, then that is the unique stable solution; iii) When men propose first, the solution is optimal for the men in the sense that there is no other stable solution in which any man does better. Similarly, the version in which women propose first, results in a solution that is optimal for women.

The two Gale/Shapley algorithms can only give a solution in which men are treated very well, or a solution in which women are treated very well. What about a solution in which everyone is treated “moderately well”? Vande Vate(1989) showed that it is possible to formulate the stable marriage assignment problem as a linear program. The key observation is: if we consider any man i and woman j in

a stable solution, then one of the following must hold: a) i and j are assigned to each other, or b) man i is assigned to some other woman k whom he prefers to j , or c) woman j is assigned to some man k whom she prefers to i . If none of (a), (b), and (c) hold, then i and j both prefer each other to their current mates and they are tempted to elope.

Define the parameters and sets:

$mref_{ij}$ = the preference position of woman j for man i ,
e.g., if man 2's first choice is woman 3, then $mref_{23} = 1$,

$wref_{ij}$ = the preference position of man i for woman j ,
e.g., if woman 3's second choice is man 1, then $wref_{13} = 2$,

$SM(i,j)$ = the set of women that man i prefers to woman j ,
 $= \{ k : mref_{ik} < mref_{ij} \}$,

$SW(i,j)$ = the set of men that woman j prefers to man i ,
 $= \{ k : mref_{kj} < mref_{ij} \}$,

Define the variables:

$y_{ij} = 1$ if man i and woman j are assigned to each other.

The "no eloping" stability conditions (a), (b), and (c) above correspond to the linear constraints:

For all men i and women j :

$$y_{ij} + \sum_{k \in SM(i,j)} y_{ik} + \sum_{k \in SW(k,j)} y_{kj}.$$

A remaining question is, what objective function should we use? We already saw a solution above in which men were treated well but women were treated poorly, and a solution in which women were treated well but men were treated poorly. How about a solution in which minimizes the worst that anyone gets treated? The following LINGO model illustrates.

```
! Stable Marriage Assignment(stable_marriage3);
SETS:
  MAN: AM;
  WOMAN: AW;
  MXW(MAN,WOMAN): MPREF, WPREF, Y, RM, RW;
ENDSETS
DATA:
! Example from Gale and Shapley(1962);
  MAN = ADAM BOB CHUCK;
  WOMAN= ALICE BARB CARMEN;
! Men(row) preference for women(col);
  MPREF =
    1      2      3 !ADAM;
    3      1      2 !BOB;
    2      3      1;!CHUCK;
! Women(col) preference for men(row);
  WPREF =
    3      2      1 !ADAM;
    1      3      2 !BOB;
    2      1      3;!CHUCK;
```

```

! Thus, Adam's first choice is Alice.
    Alice's first choice is Bob;
! This data set has 3 stable assignments;
ENDDATA
! Y(i,j) = 1 if man i is assigned to woman j;

! Each man must be assigned;
@FOR(MAN(i):
    @SUM(WOMAN(j): Y(i,j)) = 1;
);

! Each woman must be assigned;
@FOR(WOMAN(j):
    @SUM(MAN(i): Y(i,j)) = 1;
);

! Stability conditions: Either man i and woman are
assigned to each other, or
man i gets a woman k he prefers to j, or
woman j, gets a man k she prefers to i;
@FOR( MXW(i,j):
    Y(i,j)
    + @SUM(WOMAN(k) | MPREF(i,k) #LT# MPREF(i,j): Y(i,k))
    + @SUM( MAN(k) | WPREF(k,j) #LT# WPREF(i,j): Y(k,j)) >= 1
);

! Compute actual assigned rank for each man and woman;
@FOR( MAN(i):
    AM(i) = @SUM( WOMAN(k): MPREF(i,k)*Y(i,k));
    PWORST >= AM(i);
);
@FOR(WOMAN(j):
    AW(j) = @SUM( MAN(k): WPREF(k,j)*Y(k,j));
    PWORST >= AW(j);
);

! Minimize the worst given to anyone;
MIN = PWORST;

```

When solved, we get the solution:

Variable	Value
Y(ADAM, BARB)	1.000000
Y(BOB, CARMEN)	1.000000
Y(CHUCK, ALICE)	1.000000

In the “Men first” solution, every woman got her third choice. In the “Woman first” solution, every man got his third choice. In this solution, the worst anyone gets is their second choice. In fact, everyone gets their second choice. McVitie and Wilson(1971) present an algorithm for efficiently enumerating all stable solutions.

For this example, we have an answer to question (d) above. It is easy to see that each of the three solutions is Pareto optimal. In the “Women first” solution, clearly the women cannot do any better,

and the men cannot do any better without hurting one of the women. Similar comments apply to the other two solutions.

With regard to the incentive compatibility question, (e) above, Roth, Rothblum, and Vande Vate provide a partial answer, namely, if the “Men first” algorithm is used then there is nothing to be gained by a man misrepresenting his preferences. This is somewhat intuitive in that if the “Men first” rule is used, then the resulting solution gives each man the best solution possible among all stable solutions. We may reasonably restrict ourselves to stable solutions. Thus, if some man misrepresents his preferences, this might cause a different stable solution to result in which this man might be treated worse, but definitely no better. Abdulkadiroglu, Pathak, and Roth(2005), mention that New York City, when assigning students to highschools, uses a “Students first” variant of the marriage assignment algorithm so as to motivate students to state their true preferences among highschools they are considering attending.

16.5.1 The Stable Room-mate Matching Problem

The stable room-mate problem is the multi-person interpretation of the 2-matching optimization problem. A college wants to match incoming freshman, two to a room in a freshman dormitory. Each student provides a ranking of all other potential room-mates. A matching is stable if there are no two students, i and j , who are not room-mates such that i prefers j to his current room-mate, and j prefers i to his current room-mate. The stable marriage problem can be interpreted as a special case of the room-mate matching problem in which people give very unattractive rankings to people of the same sex.

In contrast to the stable marriage problem, there need not be a stable solution to a stable room-mate problem. The following 4-person example due to Gale and Shapley(1962) illustrates a situation with no stable matching.

```

! Example from Gale and Shapley;
PERSON = AL    BOB    CAL DON;
! Row preference for col;
PREF =
  99  1  2  3
  2  99  1  3
  1  2  99  3
  1  2  3  99;
! E.g., AL
! The 99 is to indicate that a person cannot be
  matched to himself.

```

Consider, for example, the solution: AL with BOB, and CAL with DON. It is not stable because BOB is matched with his second choice and CAL is matched with his third choice, whereas if BOB and CAL got together, BOB would get his first choice and CAL would get his second choice. That would give us the solution BOB with CAL, and AL with DON. This solution is not stable, however, because then AL and CAL would discover that they could improve their lot by getting together to give: AL with CAL, and BOB with DON. This solution is not stable, etc. In the terminology of game theory, the marriage assignment problem always has a core. The room-mate matching problem may not have a core.

Irving(1985) gives an efficient algorithm for detecting whether a room-mates problem has a stable matching, and if yes, finding a stable matching. The room-mates problem can also be solved by formulating it as a mathematical program as illustrated by the following LINGO model for finding a stable room-mate matching among 8 potential room-mates. This example from Irving(1985) has three stable matchings.

```

! Stable Roommate Matching(stable_roommate8);
! Each of 2n people specify a rank, 1, 2,..., 2n-1, for
each other person. We want to pair up the people into
a stable set of pairs, i.e., there are no two people
i and j who are not paired up, but would prefer to be
paired up rather than be paired with their current partner.
It may be that there is no such a stable pairing. This
LINGO model will find such a pairing if one exists, and
will minimize the worst that any person gets treated under
this pairing.

SETS:
PERSON: AP;
PXP(PERSON,PERSON): PREF, Y, R, NOSTAB;
ENDSETS

DATA:
! Example from Irving(1985);
PERSON = 1..8;
! Row preference for col;
PREF!=1 2 3 4 5 6 7 8;
 99 1 7 3 2 4 5 6
 3 99 1 7 6 2 4 5
 7 3 99 1 5 6 2 4
 1 7 3 99 4 5 6 2
 2 4 5 6 99 1 7 3
 6 2 4 5 3 99 1 7
 5 6 2 4 7 3 99 1
 4 5 6 2 1 6 3 99;
! E.g., the first choice of 1 is 2. The first choice
of 8 is 5.
! The 99 is to indicate that a person cannot be
matched to himself.
! This data set has 3 stable matchings;
ENDDATA

! Y(i,j) = 1 if PERSON i and j are matched, for i < j;

NP = @SIZE(PERSON);
! Each person must be assigned;
@FOR(PERSON(i):
  @SUM(PERSON(k)| k #GT# i: Y(i,k))
+ @SUM(PERSON(k)| k #LT# i: Y(k,i)) = 1;
);

! Turn off the lower diagonal part of Y;
@SUM( PXP(i,j)| i #GT# j: Y(i,j)) = 0;

! Enforce monogamy by making the Y(i,j) = 0 or 1;
@FOR( PXP(i,j):
  @BIN(Y(i,j))
);

! Stability conditions: Either person i and person j
are assigned to each other, or

```

```

person i gets a person k he prefers to j, or
person j gets a person k he prefers to i, or
there is no stable solution;
@FOR( PXP(i,j) | i #LT# j:
    Y(i,j)
    +@SUM(PERSON(k) | k #LT# i #AND# PREF(i,k) #LT# PREF(i,j): Y(k,i))
    +@SUM(PERSON(k) | k #GT# i #AND# PREF(i,k) #LT# PREF(i,j): Y(i,k))
    +@SUM(PERSON(k) | k #LT# j #AND# PREF(j,k) #LT# PREF(j,i): Y(k,j))
    +@SUM(PERSON(k) | k #GT# j #AND# PREF(j,k) #LT# PREF(j,i): Y(j,k))
    + NOSTAB(i,j) >= 1
);

! Compute actual assigned rank for each person;
@FOR( PERSON(i):
    AP(i) = @SUM( PERSON(k) | i #LT# k: PREF(i,k)*Y(i,k))
    + @SUM( PERSON(k) | k #LT# i: PREF(i,k)*Y(k,i));
    PWORST >= AP(i);
);

! Compute number of instabilities;
NUMUSTAB = @SUM(PXP(i,j): NOSTAB(i,j));
! Apply most weight to getting a stable solution;
MIN = NP*NP*NUMUSTAB + PWORST;

```

Notice in the resulting solution below, there is a stable matching, i.e. $\text{NUMUSTAB} = 0$, and, no participant received worse than his second choice.

Variable	Value
NUMUSTAB	0.000000
Y(1, 5)	1.000000
Y(2, 6)	1.000000
Y(3, 7)	1.000000
Y(4, 8)	1.000000

16.6 Should We Behave Non-Optimally to Obtain Information?

One of the arts of modeling is knowing which details to leave out of the model. Unfortunately, the most likely details left out of a model are the things that are difficult to quantify. One kind of difficult-to-quantify feature is the value of information. There are a number of situations where, if value of information is considered, then one may wish to behave non-optimally, at least in the short run. Three situations to consider are: 1) We would like to gain information about a customer or supplier, e.g., a more precise description of the customer's demand curve or credit-worthiness, 2) We do not want to communicate too much information to a competitor, or 3) We want to communicate information to a business partner, e.g., a supplier.

As an example of (1) suppose we extend credit to some customers. If our initial credit optimization model says “never extend credit to customers with profile X”, then we may nevertheless wish to occasionally extend credit to such customers in order to have up-to-date information of the credit worthiness of customers with profile X. In the inventory setting where unsatisfied demand is lost and not observed, Ding and Puterman(2002) suggest that it may be worthwhile to stock a little more than “optimal” so as to get a better estimate of customer demand.

Regarding (2), we may wish to behave non-optimally so as to not reveal too much information. Any good poker player knows that one should occasionally bluff by placing a large bet, even though the odds associated with the current hand do not justify a large bet. If other players know you never bluff, then they will drop out early and not give you the chance of winning large bets, any time you make a large bet. Similarly, there was a rumor at the end of World War II that Britain allowed a bombing attack on Coventry on one occasion even though Britain knew in advance of the attack, thanks to its code-breaking. The argument was that if Britain had sent up a large fleet of fighter in advance to meet the incoming German bombers, the Germans would have known, earlier than Britain desired that Britain had broken the German communications code.

An example of (3) comes from inventory control. An optimal inventory model may recommend using a very large order size. If we use a smaller order size, however, we will be giving more timely information to our supplier about retail demand for his product. In between orders, the supplier has no additional information about how his product is selling. In the extreme, if we used an order size of 1, then the supplier would have very up-to-date information about retail demand and could do better planning.

In probability theory there is a problem class known as the multi-armed bandit problem that is similar to case (1). A decision maker (DM) must decide which one of several slot machines (one armed bandits) should be selected for the next bet. The DM strongly suspects that the expected payoff is different for different machines. From a simple pure optimization perspective, the DM would bet only on the machine with the highest expected payoff. From an information perspective, however, the DM wants to scatter the bets a little bit in order to better estimate the expected payoff of each machine. This trade-off between optimization vs. experimentation is sometimes called the explore vs. exploit decision.

16.7 Problems

- Both Big Blue, Inc. and Golden Apple, Inc. are “market oriented” companies and feel market share is everything. The two of them have 100% of the market for a certain industrial product. Blue and Gold are now planning the marketing campaigns for the upcoming selling season. Each company has three alternative marketing strategies available for the season. Gold’s market share as a function of both the Blue and Gold decisions are tabulated below:

**Payment To Blue by Gold as a Function
of Both Decisions**

		Blue Decision		
		A	B	C
		X	.4	.6
Gold Decision	Y	.3	.7	.4
	Z	.5	.9	.5

Both Blue and Gold know the above matrix applies. Each must make their decision before learning the decision of the other. There are no other considerations.

- What decision do you recommend for Gold?
- What decision do you recommend for Blue?

2. Formulate an LP for finding the optimal policies for Blue and Gold when confronted with the following game:

Payment To Blue By Gold as a Function of Both Decisions

		Blue Decision			
		A	B	C	D
Gold Decision	X	2	-2	1	6
	Y	-1	4	5	-1

3. Two competing manufacturing firms are contemplating their advertising options for the upcoming season. The profits for each firm as a function of the actions of both firms are shown below. Both firms know this table:

Profit Contributions
Fulcher Fasteners

		Option A	Option B	Option C
Repicky	Option Y	4	8	6
	10	4	6	
Rivets	Option X	8	12	10
	8	2	4	

- a) Which pair of actions is most profitable for the pair?
- b) Which pairs of actions are stable?
- c) Presuming side payments are legal, how much would which firm have to pay the other firm in order to convince them to stick with the most profitable pair of actions?
4. The three neighboring communities of Parched, Cactus and Tombstone are located in the desert and are analyzing their options for improving their water supplies. An aqueduct to the mountains would satisfy all their needs and cost in total \$730,000. Alternatively, Parched and Cactus could dig and share an artesian well of sufficient capacity, which would cost \$580,000. A similar option for Cactus and Tombstone would cost \$500,000. Parched, Cactus and Tombstone could each individually distribute shallow wells over their respective surface areas to satisfy their needs for respective costs of \$300,000, \$350,000 and \$250,000.

Formulate and solve a simple LP for finding a plausible way of allocating the \$730,000 cost of an aqueduct among the three communities.

5. Sportcasters say Team I is out of the running if the number of games already won by I plus the number of remaining games for Team I is less than the games already won by the league leader. It is frequently the case that a team is mathematically out of the running even before that point is reached. By Team I being mathematically out of the running, we mean there is no combination of wins and losses for the remaining games in the season such that Team I could end the season having won more games than any other team. A third-place team might find itself mathematically though not obviously out of the running if the first and second place teams have all their remaining games against each other.

Formulate a linear program that will not have a feasible solution if Team I is no longer in the running.

The following variables may be of interest:

x_{jk} = number of times Team j may beat Team k in the season's remaining games and Team I still win more games than anyone else.

The following constants should be used:

R_{jk} = number of remaining games between Team j and Team k . Note the number of times j beats k plus the number of times k beats j must equal R_{jk} .

T_k = total number of games won by Team k to date. Thus, the number of games won at season's end by Team k is T_k plus the number of times it beat other teams.

6. In the 1983 NBA basketball draft, two teams were tied for having the first draft pick, the reason being that they had equally dismal records the previous year. The tie was resolved by two flips of a coin. Houston was given the opportunity to call the first flip. Houston called it correctly and therefore was eligible to call the second flip. Houston also called the second flip correctly and thereby won the right to negotiate with the top-ranked college star, Ralph Sampson. Suppose you are in a similar two-flip situation. You suspect the special coin used may be biased, but you have no idea which way. If you are given the opportunity to call the first flip, should you definitely accept, be indifferent, or definitely reject the opportunity (and let the other team call the first flip). State your assumptions explicitly.
7. A recent auction for a farm described it as consisting of two tracts as follows:

Tract 1: 40 acres, all tillable, good drainage.

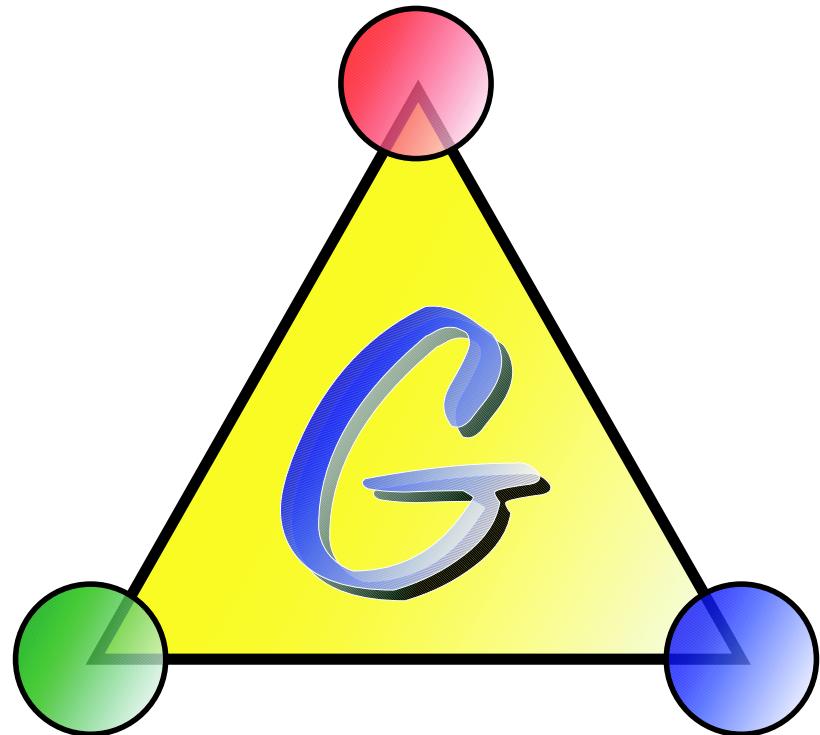
Tract 2: 35 acres, of which 30 acres are tillable, 5 acres containing pasture, drainage ditch and small pond.

The format of the auction was described as follows. First Tract 1 and Tract 2 will each be auctioned individually. Upon completion of bidding on Tract 1 and Tract 2, there will be a 15 minute intermission. After that period of time, this property will be put together as one tract of farmland. There will be a premium added to the total dollar price of Tract 1 and Tract 2. This total dollar amount will be the starting price of the 75 acres. If, at that time, no one bids, then the property will go to the highest bidders on Tracts 1 and 2. Otherwise, if the bid increases, then it will be sold as one.

Can you think of some modest changes in the auction procedure that might increase the total amount received for the seller? What are some of the game theory issues facing the individual bidders in this case?

GIDEN

a graphical environment for network optimization



VERSION J-2.0 β
JUNE 26, 2003

Collette R. Coullard
David S. Dilworth
Jonathan H. Owen

Java™ and all Java-based trademarks and logos
are trademarks or registered trademarks of Sun
Microsystems, Inc. in the United States and other
countries.

Windows® is a registered trademark of Microsoft®, Inc.

Solaris® is a registered trademark of Sun Microsystems, Inc.

Netscape® is a registered trademark of Netscape Communications Corporation.

About the Authors

Collette R. Couillard is an Associate Professor and a Charles Deering McCormick Professor of Teaching Excellence at Northwestern University. Her research focuses on modeling, solution techniques, and the fundamental mathematics of combinatorial optimization. She obtained her Ph.D. in 1985 from Northwestern University; she has taught at Purdue University and the University of Waterloo, and she was a Humboldt Fellow at the University of Bonn.

David S. Dilworth is the principal of Systems Research, a small business that develops advanced computing systems for industry and academia. His research area is electronic holography for imaging through scattering media, and he holds a patent for that work. He obtained his Ph.D. in 1989 from the University of Michigan where he is currently an Adjunct Associate Professor in Electrical Engineering and Computer Science.

Jonathan H. Owen is a research engineer in the Enterprise Systems Lab at General Motors. In 1998 he received his Ph.D. from the Industrial Engineering and Management Sciences Department at Northwestern University, where he maintains a part-time position as an assistant research professor. His primary research focuses on solving mixed-integer linear programs with general-integer variables. Additional research has dealt with solving specialized binary integer programs using branch-and-cut methodology and the development of GIDEN as a teaching and research tool for network optimization.

Resources

The latest information on the GIDEN environment is available through the web site, which is located at the following url:

<http://giden.nwu.edu/>

GIDEN is under ongoing development. If you have feedback on the environment or any of the associated documentation, please contact us via email at giden@iems.nwu.edu or through the web site.

Acknowledgments

The developers gratefully acknowledge essential support for the GIDEN project from the following organizations:

Office of Naval Research	http://www.onr.navy.mil/
Sun Microsystems, Inc.	http://www.sun.com/
National Science Foundation	http://www.nsf.gov/
Optimization Technology Center	http://www.mcs.anl.gov/otc/
Committee on Institutional Cooperation	http://NTX2.cso.uiuc.edu/cic/
Zero G Software, Inc	http://www.zerog.com/

The authors would also like to express their appreciation to those individuals who have provided comments and suggestions that have resulted in the improvement of GIDEN, including undergraduate and graduate students at Northwestern University who have taken IEMS courses C-13, D-50, and D-52. We are particularly grateful to the following individuals: Robert Fourer, Peh Ng, Susan Owen, and Donald Wagner.

Special thanks are offered to Susan Owen for her many contributions to the GIDEN project.

3 Solver Reference	33
3.1 Minimum Spanning Tree Problem	35
3.1.1 Kruskal's Algorithm	36
3.1.2 Prim's Algorithm	37
3.2 Shortest Path Problem	39
3.2.1 Dijkstra's Algorithm	40
3.2.2 FIFO Label Correcting Algorithm	42
3.3 Maximum Flow Problem	44
3.3.1 Generic Augmenting Path Algorithm	45
3.3.2 Shortest Augmenting Path Algorithm	46
3.3.3 Preflow-Push Algorithm	48
3.4 Minimum Cost Flow Problem	50
3.4.1 Capacity Scaling	51
3.4.2 Cycle Canceling	53
3.4.3 Out-of-Kilter	54
3.4.4 Network Simplex Algorithm	56
3.4.5 Successive Shortest Paths Algorithm	58
4 Developing Solvers	61
4.1 GIDEN Solver Basics	61
4.1.1 The Implementor Distribution	61
4.1.2 Anatomy of a Solver	61
4.1.3 The Obligatory HelloWorld Example	64
4.1.4 Linking a Solver to GIDEN	65
4.2 An Example Solver	65
4.2.1 Kruskal's Algorithm	66
4.2.2 Creating the Solver File and Linking to GIDEN	66
4.2.3 Adding the Algorithm Logic	67
4.2.4 Setting the Network Labels	70
4.2.5 Adding Animation	71
4.2.6 Adding Pseudocode	74
4.2.7 Using the Status Line	76

4.3 Advanced Topics	78
4.3.1 Visually Displaying Results	78
4.3.2 Using a Solver Input Dialog	81
4.3.3 Executing Solvers as Subroutines	84
4.3.4 Modifying Networks within Solvers	84
Bibliography	85

Introduction

Let $G = (N, A)$ be a graph with node set N and edge set A . Given information I about the nodes and edges of G , we call $\mathcal{N} = (G, I)$ a *network*. Networks are useful mathematical modeling constructs for representing a variety of physical and abstract systems. Given some system that can be modeled as a network \mathcal{N} , we may ask questions about what properties characterize the system. These questions can be translated into problems that we can solve using the network model \mathcal{N} . Network optimization deals with constructing a network to model an underlying system, describing in terms of the network model the properties of interest, solving the network problem using exact or heuristic techniques, and using the model solution to develop a better understanding of the real-world system. Examples of network optimization problems include shortest path problems, minimum spanning tree problems, flow problems, and matching problems. Two recent textbooks on network optimization are [1] and [3].

GIDEN is a visually oriented interactive software environment for network optimization. There are many accounts of situations where visual representations have been used effectively to communicate ideas when verbal and textual representations alone have been less successful. For network optimization, the use of *visualization* is natural because of the convenient graphical representation of nodes as points and edges as lines between pairs of nodes. (For a nice reference on visualization and optimization, see [5].) The fundamental purpose of GIDEN is to facilitate the visualization of network optimization problems, solutions, and solution algorithms. Features of GIDEN include the following:

- a graphical interface for building and modifying networks
- facilities for algorithm animation
- an expandable toolkit of animated solution algorithm implementations (“solvers”)
- an extensive library of network-related data structures
- platform independence

The features listed above make GIDEN an attractive tool for a variety of users. The most obvious users in the context of viewing algorithms are in the academic community. Teachers of network optimization can use the graphical animation tool in the classroom to demonstrate network algorithms. Algorithms are learned through various methods, depending on needs and learning styles. Typically at some stage in the learning process the steps of the algorithm are performed on an example. An effective example illustrates the steps of the procedure, including termination, and also provides some convincing evidence of the correctness of the final solution. A graphical animation tool enables users to make this stage of the learning process as extensive as they want. They can try the algorithm on a number of examples or on a specific example a number of times. They can test their knowledge of the algorithm by predicting its behavior. Students learning network optimization can use the tool outside of class to solidify their knowledge, both through independent endeavors and

through specific homework assignments. GIDEN has been well-received when used as a teaching and learning aid in graduate and undergraduate classes at Northwestern University.

GIDEN is also a useful implementation environment for network optimization algorithms. Users can quickly implement network optimization algorithms using the network data structures class library, which provides abstract data types and containers that are appropriate for network programming (e.g., `Node`, `Edge`, `Network`, `List`, `PriorityQueue`). Algorithm animation assists in implementation debugging as well as identification of bottlenecks. A pseudocode window may optionally be created with each algorithm to assist in tracing algorithm execution.

Chapter 1

Preliminaries: Network Optimization and GIDEN

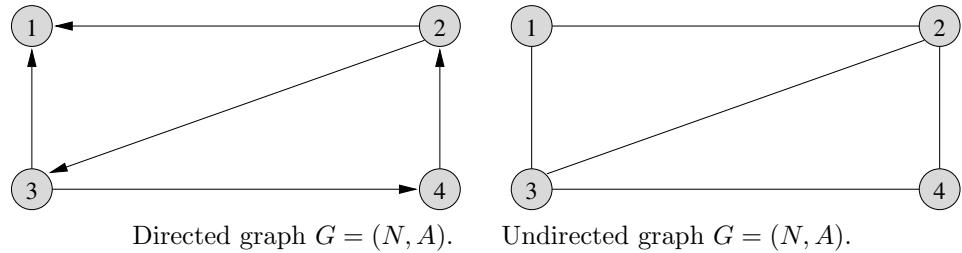
1.1 Notation and Definitions

In this section, we define notation and elementary network concepts that will be used throughout this manual. For a thorough introduction to networks, see [1], [2], [3], and [6].

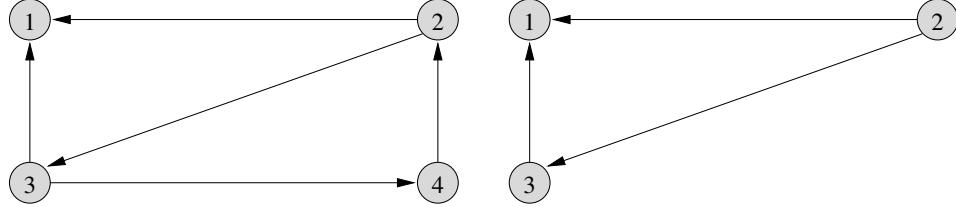
graph: A *graph* $G = (N, A)$ consists of a set N of *nodes* (or *vertices*) and a set A of *edges* (or *arcs*).

directed and undirected edges: An edge $e = (i, j)$ is a pairing of nodes i and j ; we call node i and node j the *ends* of edge e . An edge e is *directed* if it is an ordered pairing of nodes, i.e., $e = (i, j) \neq (j, i)$; if an edge $e = (i, j)$ is a directed edge, we call i the *tail node* (or *tail*) of edge e , and we call j the *head node* (or *head*) of edge e . If an edge e is an unordered pairing of nodes, i.e., $e = (i, j) = (j, i)$, then we call the edge *undirected*. A directed edge is also called an *arc*.

directed and undirected graphs: Graph G is a *directed graph* (or *digraph*) if edges $e \in A$ are directed. G is an *undirected graph* if edges $e \in A$ are undirected. For the sake of simplicity, we assume that edges in a graph are either all directed or all undirected. The illustration below, shows a graph G with node set $N = \{1, 2, 3, 4\}$ and edge set $A = \{(2, 1), (2, 3), (3, 1), (3, 4), (4, 2)\}$.

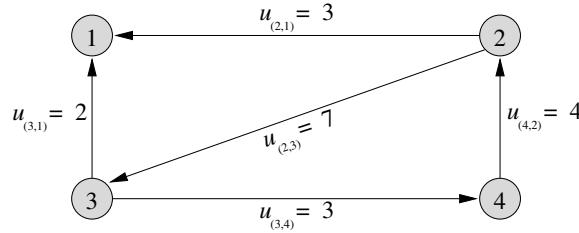


subgraph: A graph $\bar{G} = (\bar{N}, \bar{A})$ is a *subgraph* of $G = (N, A)$ if $\bar{N} \subseteq N$ and $\bar{A} \subseteq A$. In the illustration below we show a directed graph $G = (N, A)$ (left) and a subgraph $\bar{G} = (\bar{N}, \bar{A})$, where $\bar{N} = \{1, 2, 3\}$ and $\bar{A} = \{(2, 1), (2, 3), (3, 1)\}$ (right).



Given $E \subseteq A$, the subgraph *induced by* E is denoted $G(E) = (N(E), E)$, where $N(E)$ is the set of end nodes of members of E . Where no confusion arises, we sometimes use E to refer to this subgraph $G(E)$.

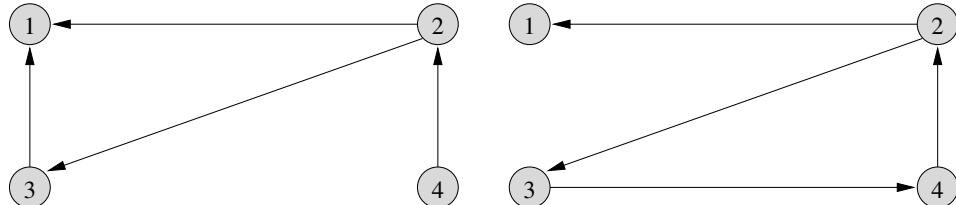
network: Given some information I about the nodes and/or edges of a graph G , we call $\mathcal{N} = (G, I)$ a *network*. If the edge set of G is directed, we call \mathcal{N} a *directed network*; otherwise, the edge set of G is undirected and we call \mathcal{N} an *undirected network*. The following illustration shows a network $\mathcal{N} = (G, u)$ where G is the directed graph from above and $u = \{u_e \in \mathbb{Z} : e \in A\}$ is some information on the edges A .



walk: A *walk* W of G is a sequence of nodes and edges

$$W = \{i_1, e_1, i_2, e_2, i_3, \dots, i_{r-1}, e_{r-1}, i_r\}$$

such that $e_k = (i_k, i_{k+1})$ or $e_k = (i_{k+1}, i_k)$ for all $k \in \{1, 2, \dots, r-1\}$. The term *walk* may also be used to refer to a sequence of either nodes or edges (e.g., the walk W can be also be written as $\{i_1, i_2, \dots, i_r\}$ or $\{e_1, e_2, \dots, e_{r-1}\}$). If G is directed, then W is called a *directed walk* if the edge $(i_k, i_{k+1}) \in A$ for any two consecutive nodes in the walk, i_k and i_{k+1} . Pictured below are two walks: walk $W_1 = \{2, 3, 1, 2, 4\}$ (left) and directed walk $W_2 = \{2, 3, 4, 2, 1\}$ (right).



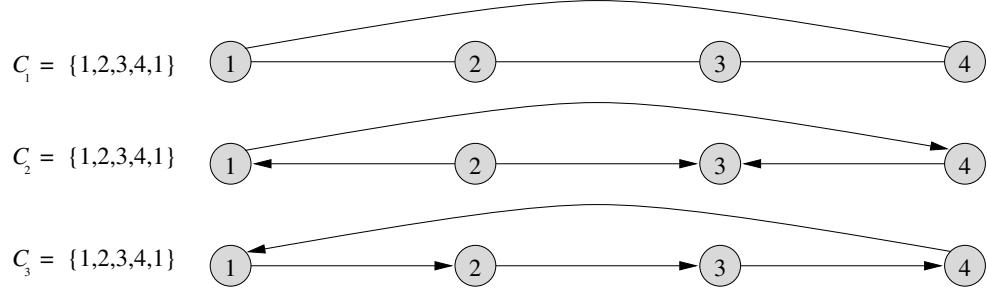
path: A *path* P is a walk with no repeated nodes. A *directed path* (or *dipath*) P is a directed walk with no repeated nodes. We call a path P an *s-t path* (*s-t dipath*) if it begins at node s and ends at node t . The following illustration shows three paths; only the bottom path, P_3 , is a directed path.

$$P_1 = \{1, 2, 3, 4\} \quad \begin{array}{ccccccc} & 1 & \longrightarrow & 2 & \longrightarrow & 3 & \longrightarrow & 4 \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \end{array}$$

$$P_2 = \{1, 2, 3, 4\} \quad \begin{array}{ccccc} & 1 & \longrightarrow & 2 & \longrightarrow \\ & \downarrow & & \downarrow & \downarrow \\ & 3 & \longleftarrow & 4 & \longleftarrow \end{array}$$

$$P_3 = \{1, 2, 3, 4\} \quad \begin{array}{ccccccc} & 1 & \longrightarrow & 2 & \longrightarrow & 3 & \longrightarrow & 4 \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow \end{array}$$

cycle: A *cycle* (or *circuit*) C is a path $\{i_1, e_1, i_2, e_2, \dots, i_{r-1}, e_{r-1}, i_r\}$ along with the edge (i_1, i_r) or the edge (i_r, i_1) . A *directed cycle* (or *directed circuit*) is a directed path followed by the directed edge (i_r, i_1) . The illustration below shows three cycles; only the bottom cycle, C_3 is a directed cycle.



acyclic: We say a graph is *acyclic* if it does not contain a cycle. Similarly, a network is called acyclic if its associated graph is acyclic.

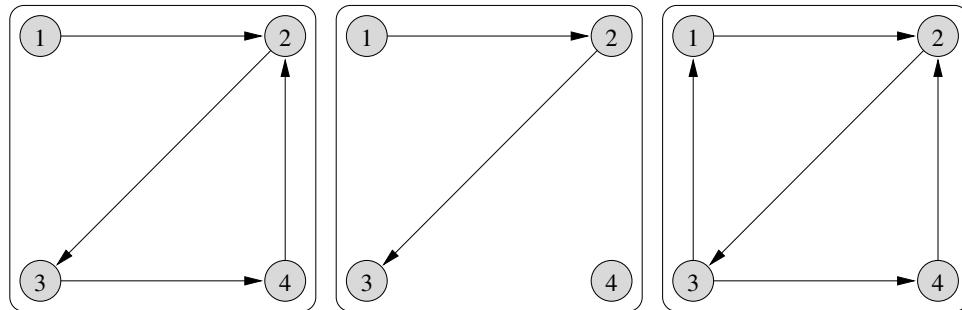
multiedge: Duplicate edges in A are called *multiedges* (or *parallel edges*). The following illustration shows a multiedge between node 1 and node 3.



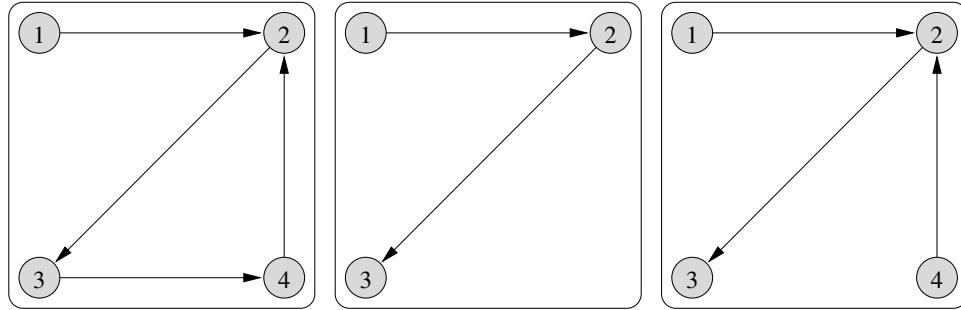
loop: An edge whose endpoints are the same node (i.e., $e = (i, i)$ for some $i \in N$) is called a *loop*. The following illustration shows a loop on node 3.



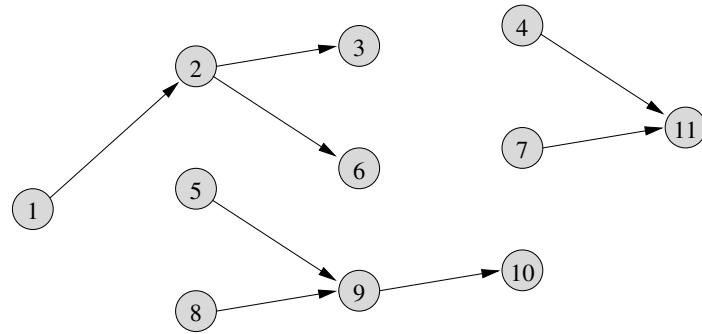
connectedness: In a graph $G = (N, A)$, two nodes $i \in N$ and $j \in N$ are called *connected* if there exists at least one i - j path in G . A *connected graph* is a graph $G = (N, A)$ where every pair of nodes in N is connected. The maximal connected subgraphs of G are called the *components* of G . If there exists a pair of nodes $i, j \in N$ such that there is no path between node i and node j , then we call G a *disconnected graph*. We say that a digraph G is *strongly connected* if for each pair of nodes, $i, j \in N$, there exists both an i - j dipath and a j - i dipath in G . In the following illustration, we show a connected graph (left), a disconnected graph (middle), and a strongly connected graph (right).



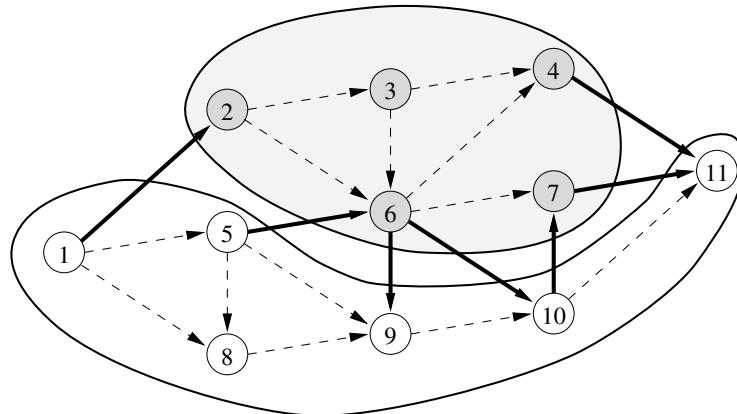
tree: A *tree* is an acyclic connected graph. A *spanning tree* of graph G is a tree T that is a subgraph of G containing all nodes of G . In the following illustration, we show a graph G (left), a tree that is a subgraph of G (middle), and a tree that is a spanning tree of G (right).



forest: A *forest* is an acyclic graph; it follows that each connected component of a forest is a tree. The following illustration shows a forest with three components, $F = \{T_1, T_2, T_3\}$, where $T_1 = \{(1, 2), (2, 3), (2, 6)\}$, $T_2 = \{(4, 11), (7, 11)\}$, and $T_3 = \{(5, 9), (8, 9), (9, 10)\}$.

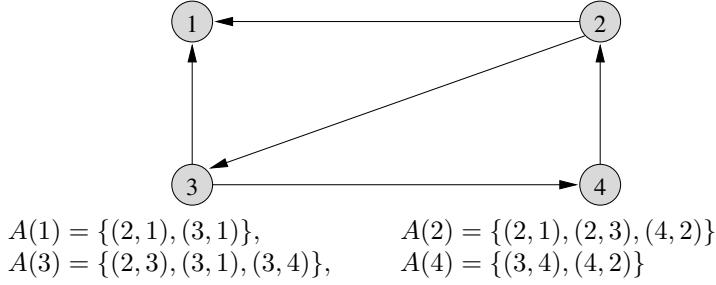


cut: Given a partition of the node set N into two subsets N_1 and $N_2 = N \setminus N_1$, the collection of edges (i, j) such that $i \in N_1$ and $j \in N_2$ is called a *cut* (or *cutset*), denoted $\delta(N_1, N_2)$. For two specified nodes, $s, t \in N$, an $s-t$ *cut* is any cut $\delta(N_1, N_2)$ with the added property that $s \in N_1$ and $t \in N_2$. In the following illustration, shaded nodes belong to N_1 and non-shaded nodes belong to N_2 ; the edges drawn as solid arrows are in the cut $\delta(N_1, N_2)$.

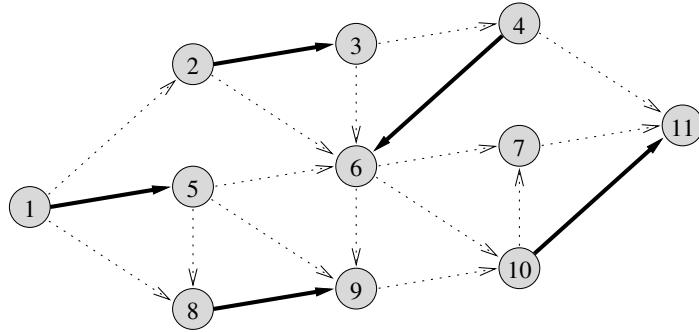


$$N_1 = \{2, 3, 4, 6, 7\}, N_2 = \{1, 5, 8, 9, 10, 11\}, \\ \delta(N_1, N_2) = \{(1, 2), (4, 11), (5, 6), (6, 9), (6, 10), (7, 11), (10, 7)\}$$

adjacency: We say that node i and node j are *adjacent nodes* if edge $(i, j) \in A$ or edge $(j, i) \in A$. For a node $i \in N$, we say that any edge $(i, j) \in A$ or $(j, i) \in A$ is an *incident edge* of node i . The collection of edges that are incident to a node $i \in N$ is called the *adjacency list* of node i ; we denote the adjacency list of node i as $A(i)$. The following illustration shows a graph and gives the adjacency list for each node.

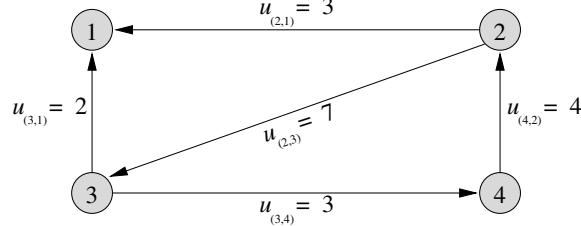


matching: A *matching* of a graph G is a subset $M \subseteq A$ such that each node of G has at most one incident edge in M . An example of a matching is shown as the bold edges in the following illustration; in this example, the matching is given as $M = \{(1, 5), (2, 3), (4, 6), (8, 9), (10, 11)\}$.



Definition: Given a finite set E and a field \mathbb{F} , \mathbb{F}^E denotes the $|E|$ -dimensional vector space with components indexed on the members of E . For example, $u \in \mathbb{Z}^A$ means $\{u_e \in \mathbb{Z} : e \in A\}$.

capacitated network: A *capacitated directed network* is a directed network $\mathcal{N} = (G, u)$ where information $u \in \mathbb{Z}^A$ consists of integer-valued edge capacities for each edge $e \in A$.

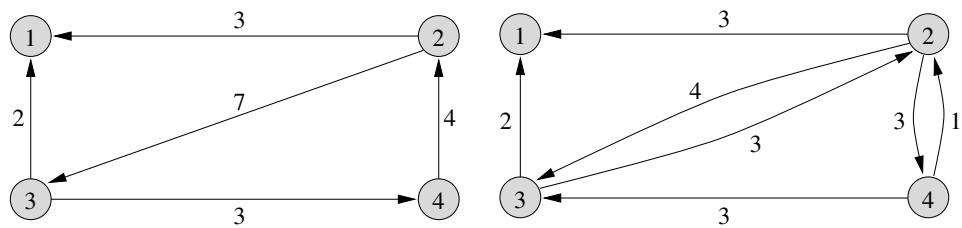


flow: Given a capacitated directed network $\mathcal{N} = (G, u)$, a *flow* is a vector $x \in \mathbb{R}^A$ that satisfies *flow conservation* at each of the nodes of the network, that is $\sum_{(i,j) \in E} x_{(i,j)} = \sum_{(j,i) \in E} x_{(j,i)}$ for each node $j \in N$. A *feasible flow* also satisfies the capacity (and lower bound) restrictions on the edges, that is $0 \leq x_e \leq u_e$ for each edge $e \in A$.

residual network: Given a capacitated directed network $\mathcal{N} = (G, u)$ and a flow $x \in \mathbb{R}^A$, the associated *residual network* is the network $\mathcal{N}^x = (G^x, u^x)$. The graph $G^x = (N, A^x)$ has node set N and edge set A^x , where for each edge $(i, j) \in A$, the edge $e = (i, j)$ is in A^x if $x_e < u_e$ and the edge $e = (j, i)$ is in A^x if $x_e > 0$. The capacities of the residual network, u^x , are defined as follows:

$$u_{(i,j)}^x = \begin{cases} u_{(i,j)} - x_{(i,j)} & \text{for } (i, j) \in A \\ x_{(j,i)} & \text{for } (j, i) \in A \end{cases}$$

The illustration below shows a network \mathcal{N} (left) and the associated residual network \mathcal{N}^x (right) for flow $x = \{x_{(2,1)}, x_{(2,3)}, x_{(3,1)}, x_{(3,4)}, x_{(4,2)}\} = \{0, 3, 0, 3, 3\}$. Edge capacities for each network (u or u^x) are given as numeric labels on the edges.



1.2 An Example of Network Optimization

In this section we demonstrate network optimization by describing how it can be used to analyze a system of interconnected one-way streets: We first build a network model of our system. We then pose a question about the system and formulate a corresponding network optimization problem. We then demonstrate a process for solving the problem, using a network optimization algorithm. Finally, we show the solution of the problem on our network and translate the solution into an answer to our original question regarding the underlying system.

1.2.1 Constructing a Network Model

As mentioned in the introduction, networks are often used to represent a physical or logical system. As an example, let's consider a system of interconnected one-way streets. In order to represent this system as a graph, let each intersection of two or more streets be represented by a node $i \in N$. Given a pair of intersections $i, j \in N$, if you can travel along a street from i to j without encountering any other intersection, let the street segment between i and j be represented as a directed edge $e = (i, j) \in A$. The system of roads is then represented by the directed graph $G = (N, A)$.

It is natural to consider visual representations of systems that can be modeled using graphs and networks, with nodes depicted as points and edges as lines between pairs of nodes. Suppose that part (a) of Figure 1.1 shows our example system of streets. In this system we have eight streets

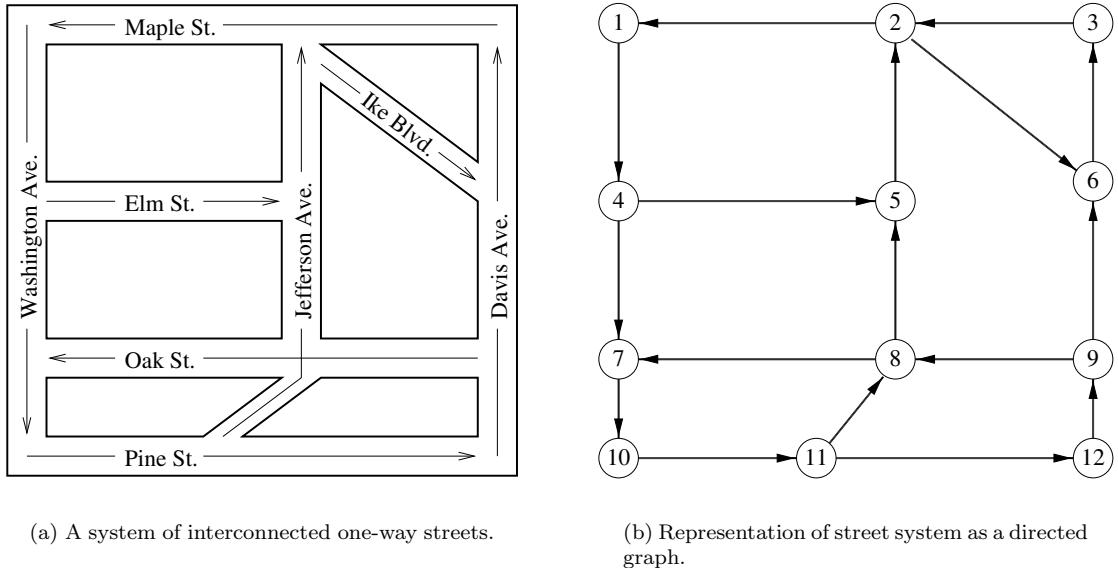


Figure 1.1: System of streets and corresponding network model.

and twelve intersections. We can represent this system as a directed graph with twelve nodes — each representing an intersection — and seventeen edges that collectively represent the eight streets. In particular, we can represent the system pictured in Figure 1.1a as a directed graph $G = (N, A)$ where

$$N = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

and

$$A = \{(1, 4), (2, 1), (2, 6), (3, 2), (4, 5), (4, 7), (5, 2), (6, 3), (7, 10), (8, 5), (8, 7), (9, 6), (9, 8), (10, 11), (11, 8), (11, 12), (12, 9)\}.$$

Part (b) of Figure 1.1 shows a representation of this directed graph.

Suppose we also have values l associated with each arc in A that represent the length of each street segment. In this case, the system of roads and associated street lengths are represented by the (directed) network $\mathcal{N} = (G, l)$. For the example system illustrated in Figure 1.1a, lengths of the street segments are given in Table 1.1. In the table, street segments are represented as an ordered

Table 1.1: Lengths for street segments between adjacent intersections.

“TAIL” INTERSECTION	“HEAD” INTERSECTION	LENGTH
Maple/Washington	Elm/Washington	30 units
Maple/Jefferson/Ike	Maple/Washington	50 units
Maple/Jefferson/Ike	Davis/Ike	47 units
Maple/Davis	Maple/Jefferson/Ike	40 units
Elm/Washington	Elm/Jefferson	50 units
Elm/Washington	Oak/Washington	20 units
Elm/Jefferson	Maple/Jefferson/Ike	30 units
Davis/Ike	Maple/Davis	25 units
Oak/Washington	Pine/Washington	15 units
Oak/Jefferson	Elm/Jefferson	20 units
Oak/Jefferson	Oak/Washington	50 units
Oak/Davis	Davis/Ike	25 units
Oak/Davis	Oak/Jefferson	40 units
Pine/Washington	Pine/Jefferson	40 units
Pine/Jefferson	Oak/Jefferson	18 units
Pine/Jefferson	Pine/Davis	50 units
Pine/Davis	Oak/Davis	15 units

pair of intersections such that we can drive from the *tail intersection* to the *head intersection* while obeying the one-way restrictions. The *lengths* listed give the distances between the tail and head intersections along the corresponding street segments. Figure 1.2 shows a network that represents our resulting system (complete with lengths l_e for each $e \in A$).

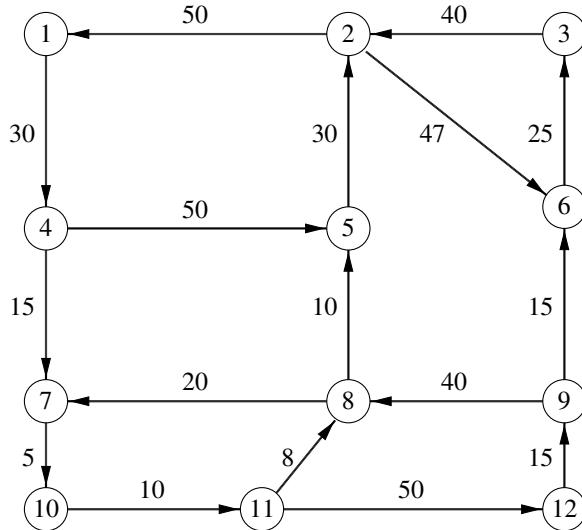


Figure 1.2: A network representing the system; street lengths are associated with each street segment.

1.2.2 Formulating a Network Optimization Problem

Given such a system of interconnected streets, we may be interested in answering questions such as

“What is the shortest driving distance between intersection-X and intersection-Y?”

This question translates into a problem that we can solve using our network.¹ Using the earlier definition of a directed path, we can translate the above question in terms of the network $\mathcal{N} = (G, l)$, where intersection-X and intersection-Y correspond to nodes $x \in N$ and $y \in N$ as follows:

“Find a shortest x - y dipath P in G .”

This problem statement leads to the following network optimization problem formulation, where the length of a dipath P is equal to the sum of the edge lengths l_e for all edges $e \in P$:

$$\begin{array}{ll} \min & \sum_{e \in P} l_e \\ \text{such that} & P \text{ is an } x\text{-}y \text{ dipath in } G \end{array}$$

We now describe how network optimization can be used to solve this problem formulation, returning a solution or some proof that no solution exists.

1.2.3 Solving a Network Optimization Problem

Methods for solving network optimization problems are known as *network algorithms*. Graph and network-based algorithms consist of a set of instructions. *Execution* of such an algorithm is the process of following these instructions by performing a series of operations on a network instance. Thus, algorithm executions result in a finite sequence of operations on the nodes, edges, and associated information that yields a solution or some proof that no solution exists. The algorithm execution process is illustrated in Figure 1.3.

Return now to our problem formulation for finding the shortest driving distance between two intersections. Consider the situation where this problem formulation does not have a solution (i.e., there is no x - y dipath P in network \mathcal{N}). In this case, we say that our problem is *infeasible*. Recall our underlying question:

What is the shortest driving distance between intersection-X and intersection-Y?

If our problem formulation is infeasible, the appropriate answer would be

You cannot drive from intersection-X to intersection-Y via the given system of streets, and thus the shortest driving distance between those two intersections is undefined (infinite).

When an x - y dipath P does exist (i.e., the problem is *feasible*), the solution to our problem is a shortest dipath with respect to the street segment lengths l . For the example system illustrated in Figure 1.1, our readers should be able to convince themselves through observation that at least one dipath exists between each pair of intersections.

Let’s consider a particular instance where we are trying to find the shortest driving distance from our house at the corner of Maple/Washington to the movie theater at Davis/Ike. One approach would be to enumerate all dipaths between the two intersections, calculate the length of each dipath, and choose a dipath with minimal total length. While this approach seems straightforward for our example system, for larger systems it becomes impractical to find all dipaths. Fortunately for us, there are network optimization algorithms that will solve our problem more efficiently, even for large networks. One such algorithm that we can apply is known as *Dijkstra’s Algorithm*.

¹In fact, what we have described is known as the *shortest-path problem*. See § 3.2 for more information on this problem.

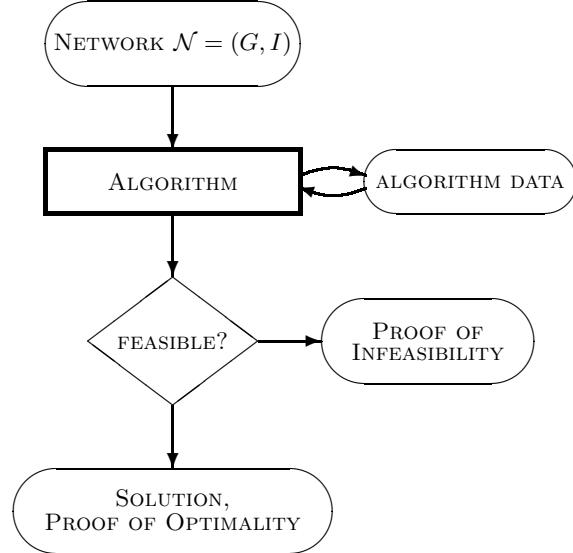


Figure 1.3: Solution process representation.

Dijkstra's Algorithm

Consider a directed network $\mathcal{N} = (G, l)$, where l_e is a non-negative length for each edge $e \in A$. Given such a network and some node $s \in N$, Dijkstra's algorithm finds the shortest dipaths from node s to all other nodes in N .² To do so, Dijkstra's algorithm maintains a partition of the node set N into three subsets: a set of *permanently labeled nodes* to which a shortest dipath is known from s , a set of *labeled nodes* to which some dipath is known from s — though it may not be the shortest such dipath available, and a set of *unlabeled nodes* to which no dipath from s is known. Figure 1.4 presents a description of Dijkstra's algorithm where these three sets are referred to as P , L , and U for permanently labeled, labeled, and unlabeled nodes respectively.

Although we do not offer a proof here, it can be shown that

1. Dijkstra's algorithm terminates finitely when given a finite network \mathcal{N} as input.
2. At completion of Dijkstra's algorithm, all nodes belong to set P or set U .
3. For each node $i \in U$ at completion of Dijkstra's algorithm, there is no dipath P in \mathcal{N} from s to i .
4. For each node $i \in P$, the shortest dipath from node s to node i has total length d_i .

Application of Dijkstra's Algorithm to Our Example

Let's now apply Dijkstra's algorithm to our example network from Figure 1.2 to find the shortest dipath from our house (node 1) to the theater (node 6). We will demonstrate the application of Dijkstra's algorithm through a series of illustrations.

²If dipaths do not exist from s to some nodes in N , the final “distance” of the shortest dipaths to those nodes will be reported as infinite.

```

algorithm Dijkstra;
{
    // initialize algorithm information
     $d_i := \infty$  and  $\text{pred}(i) := \text{null}$  for all  $i \in N$ ;
     $P := \emptyset$ ,  $L := \emptyset$ ,  $U := N$ ;

    // setup for starting at the given node  $s$ 
     $d_s := 0$ ;
    move node  $s$  from set  $U$  to set  $L$ ;

    while (the set  $L$  is non-empty) do
    {
        select node  $i \in L$  such that  $d_i \leq d_j$  for all  $j \in L$ ;
        move node  $i$  from set  $L$  to set  $P$ ;

        for (all nodes  $j$  with  $(i, j) \in E$ ) do
        {
            if ( $d_j > d_i + l_{(i,j)}$ ) then
            {
                 $d_j := d_i + l_{(i,j)}$ ;
                 $\text{pred}(j) := i$ ;
                if ( $j \in U$ ) then move  $j$  from set  $U$  to set  $L$ ;
            }
        }
    }
}

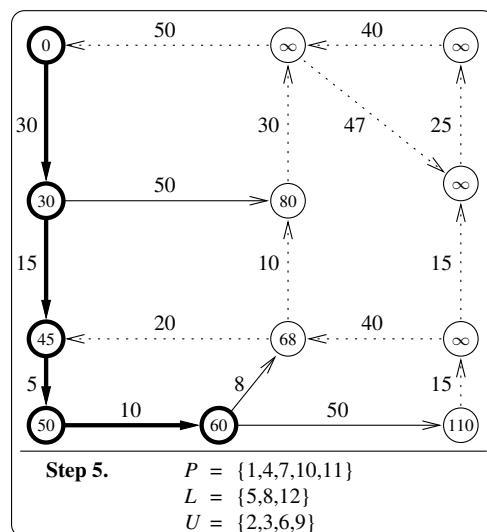
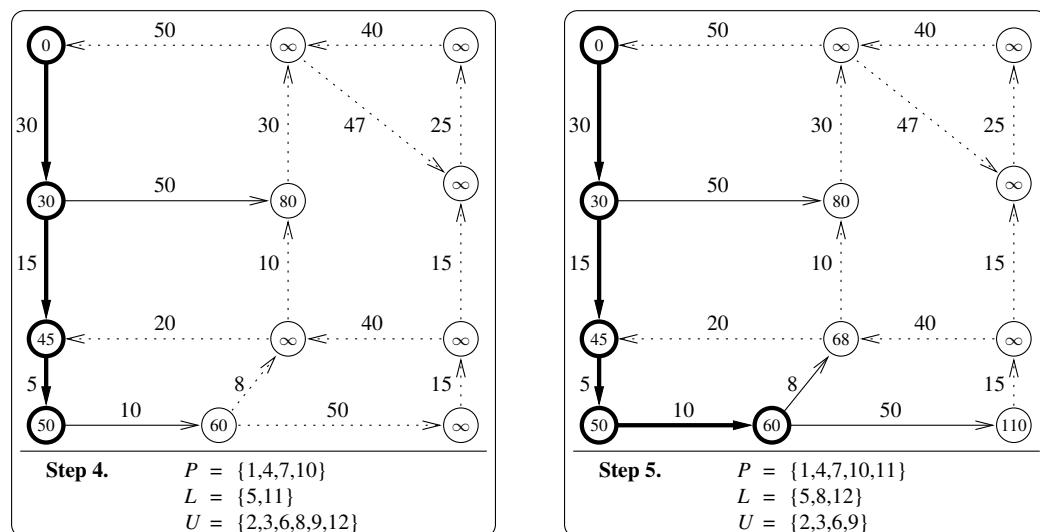
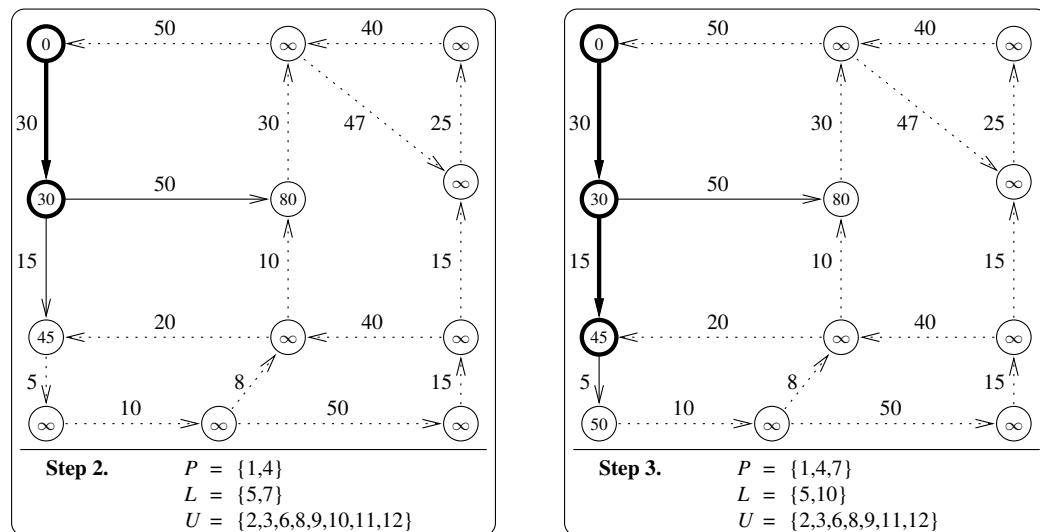
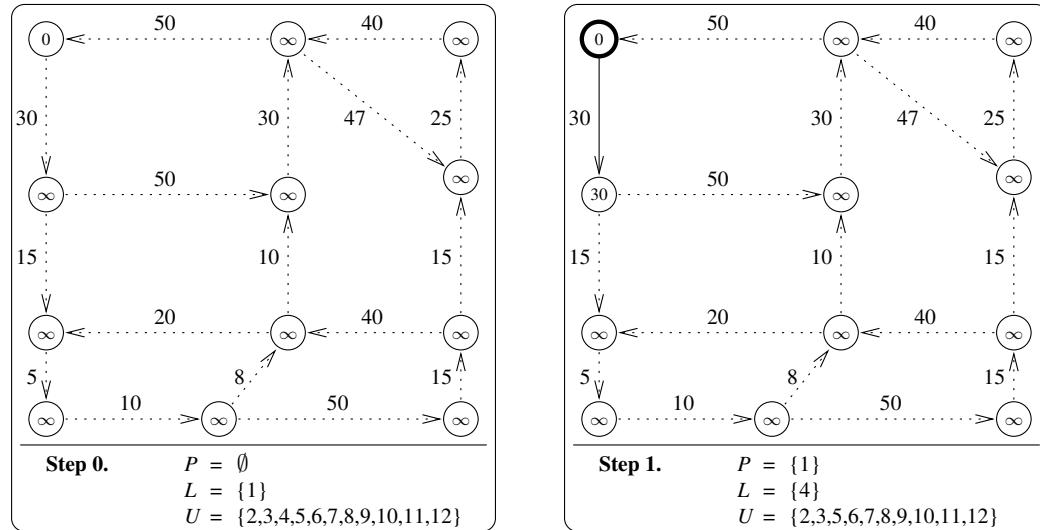
```

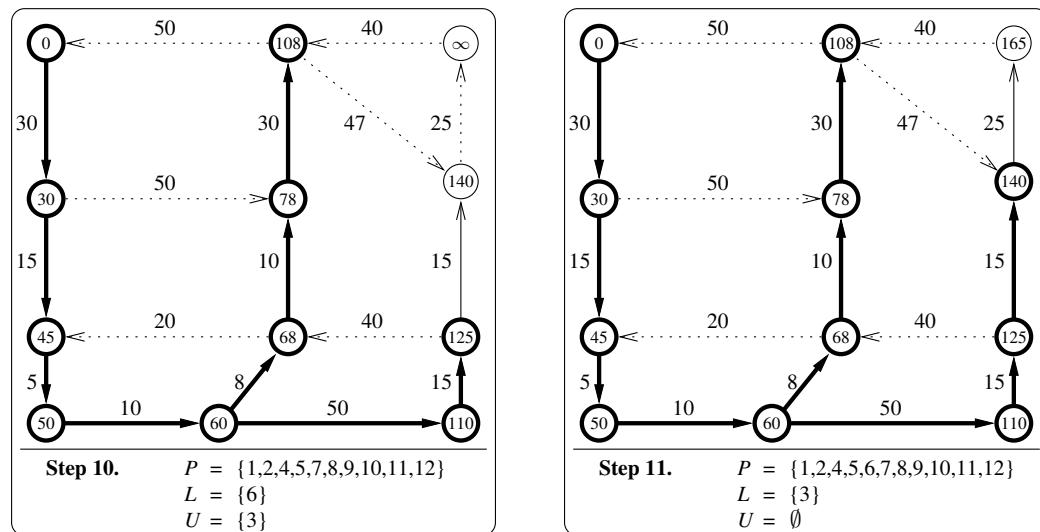
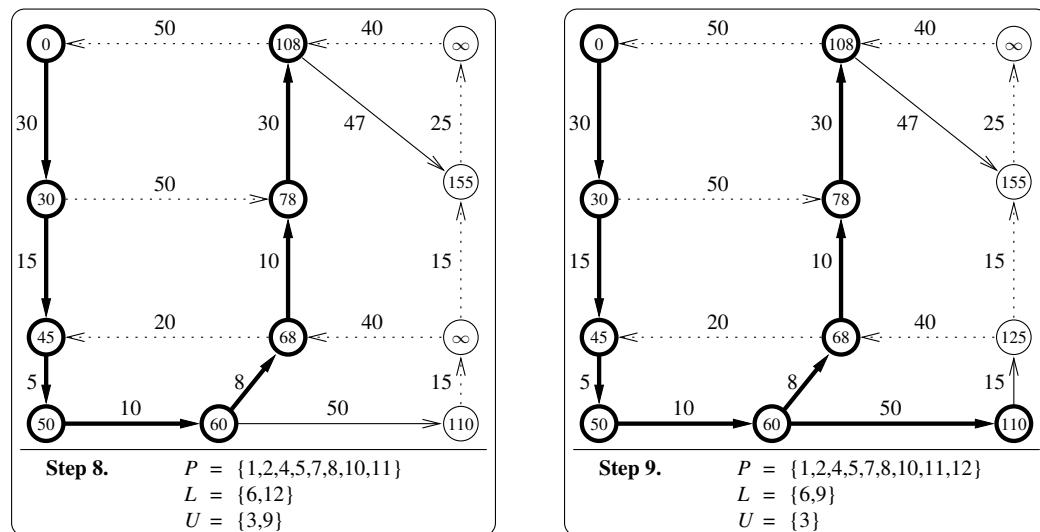
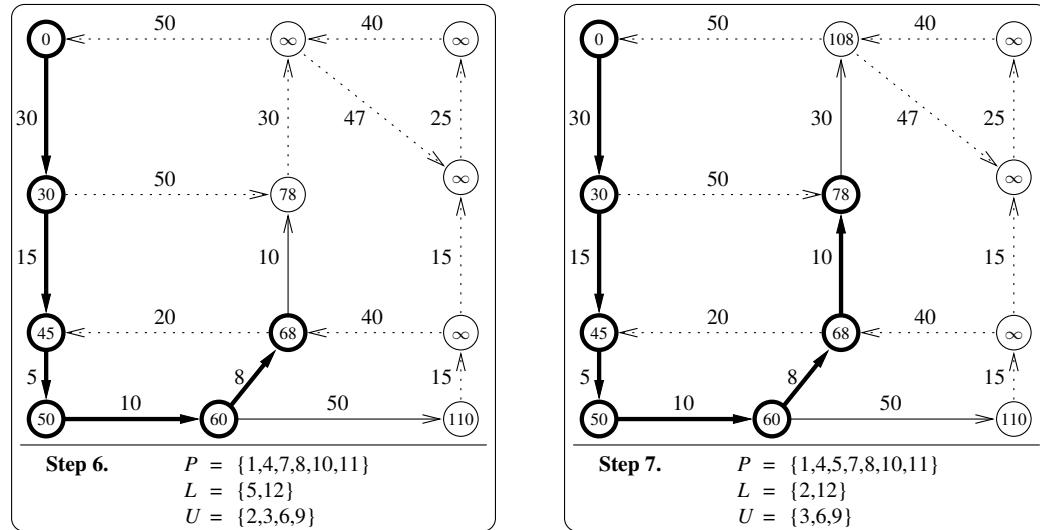
Figure 1.4: Description of Dijkstra's algorithm.

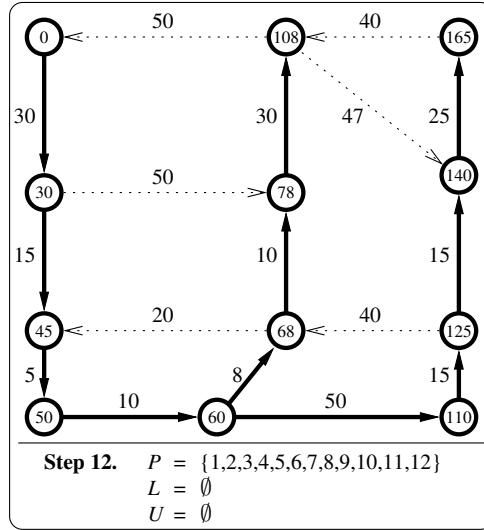
In the illustrations that follow, each node $i \in N$ is drawn with an associated distance label d_i that represents the length of the shortest dipath currently known from s to i ; for unlabeled nodes this distance label is “ ∞ .” Nodes are drawn as either solid or bold circles. Nodes drawn as solid circles may belong to either L or U , depending on whether the distance label is ∞ . Bold circles represent nodes that belong to P , i.e., those that have been permanently labeled; the distance labels for such nodes $i \in P$ gives the length of the shortest dipath from s to i .

Each edge is drawn either as a dotted arrow, a solid arrow, or a bold arrow. An edge that is drawn as a dotted arrow is not currently used in any dipath from node s . An edge $e = (i, j)$ that is drawn as a solid arrow is in the best dipath *currently known* from node s to node $j \in L$. An edge $e = (i, j)$ that is drawn as a bold arrow belongs to the best dipath from node s to node $j \in P$.

In the illustration for Step 0 we see the state of the network after algorithm information has been initialized and node s has been labeled. Step 1 – Step 12 show the state of the network after each pass through the while loop in Dijkstra's algorithm as described in Figure 1.4.







Notice that we permanently labeled node 6 in Step 11. For the purposes of our example, we would have liked to have terminated execution after this step since we are only interested in finding the shortest dipath from node s to node 6. We can specialize Dijkstra's algorithm to terminate early when it finds the shortest dipath from the start node s to a specified destination node t by changing the stopping criteria from

while the set L is non-empty **do**

to the stopping criteria

while $t \notin P$ and the set L is non-empty **do**

With this change Dijkstra's algorithm terminates execution if we permanently label the destination node t , even when L is non-empty. (Observe that by modifying the algorithm in this manner, we no longer maintain properties (2) and (3) given earlier for Dijkstra's algorithm.)

1.2.4 A Solution for a Network Optimization Problem

Throughout this section we have worked towards answering the question

What is the shortest driving distance between our house and the theater?

We have modeled the streets in our neighborhood as a network $\mathcal{N} = (G, l)$, posed the question as a network optimization problem on \mathcal{N} , and applied Dijkstra's algorithm to solve the network optimization problem. All that is left now is to interpret the solution as an answer to our question.

The solution that is given by Dijkstra's algorithm for our problem is illustrated in Figure 1.5b. For any node $i \in N$, the shortest dipath from s to i in our network is given by the set of edges that connect node s to node i . In particular, the shortest dipath from node 1 (representing our house) to node 6 (representing the movie theater) is the dipath

$$P_{1,6} = \{(1, 4), (4, 7), (7, 10), (10, 11), (11, 12), (12, 9), (9, 6)\}.$$

The total length of the dipath from node s to node i for $i \in N$ is given by the distance label d_i represented in the nodes of Figure 1.5b. For our example problem, the total length of the dipath $P_{1,6}$ between node 1 and node 6 is 140 units. Thus, we have an answer to our question:

The shortest driving distance from our house to the theater is 140 units: Drive three blocks down Washington Ave. until you reach the intersection of Washington Ave. and Pine St. Turn left onto Pine and drive two blocks until you reach Davis Ave. Turn left onto Davis and drive two blocks to the intersection of Davis and Ike Blvd.

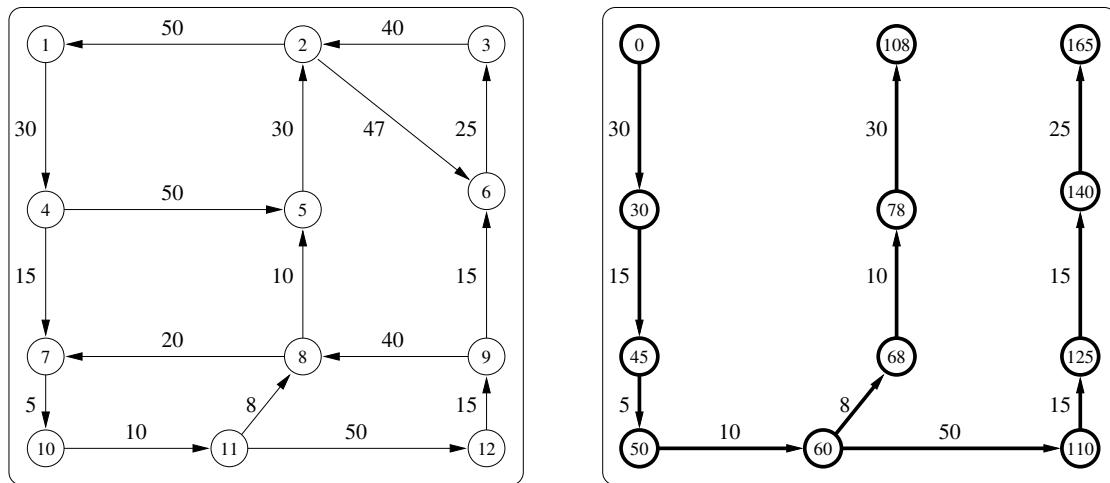


Figure 1.5: Final network and problem solution.

1.3 The GIDEN Environment

GIDEN is a visually-oriented interactive software environment for network optimization whose fundamental purpose is to facilitate the visualization of network optimization problems, solutions, and algorithms. The GIDEN system has three primary components: The first component, “core-GIDEN,” controls the graphical user interface and provides the framework for communication between solvers and graphical input/output devices. The second component is a toolkit of animated solvers. The third component is a collection of data structures that are tailored to the needs of a network optimization environment; both core-GIDEN and the solvers use these data structures extensively.

The current release of GIDEN is implemented in Java, an object-oriented programming language with built-in platform-independent graphics routines (see <http://java.sun.com>). In its current form GIDEN can be run as a stand-alone application on any Java-enabled computing platform (i.e., any machine with an available Java interpreter).³

1.3.1 Core-GIDEN

At the heart of the GIDEN environment is a framework that provides the interface between graphical input/output devices and algorithm implementations — we call this framework “core-GIDEN”. The idea behind core-GIDEN is to provide communication channels between users and solvers that are convenient for each. For users this translates into a graphical interface that is intuitive, flexible, and easy-to-use. For solvers, core-GIDEN provides convenient mechanisms for retrieving and reporting information through the graphical devices. After describing the core-GIDEN mechanism for animation, we will discuss the core-GIDEN user and solver interfaces.

Animation Sets

Solvers in GIDEN provide animation through the use of *animation sets*. Animation sets are based on the idea that the progress of a solver, given the current state of the network, can be represented by classifying nodes and edges into mutually exclusive sets. By representing each set with a distinct color, users can follow the progress of the solver by observing changes in the colors of nodes and edges. (See Chapter 3 for information on the animation sets that are used in specific solvers.)

Core Environment User Interface

Users communicate with the GIDEN environment through the graphical interface, using a mouse to select actions and the keyboard to input textual information. The graphical interface is designed to be similar to that of popular applications for the Macintosh and Windows operating systems, making the basic functionality of GIDEN easy to access and use.

GIDEN users can create and modify networks using the mouse and keyboard. Once a network is built (or opened), the user selects a solver from the solver toolkit to apply to the network. At that time, the network is analyzed and the user is prompted for any undefined information needed by the solver. The user then executes the solver, controlling the level of animation detail and the animation speed. Pseudo-code of the solution algorithm may be provided in a separate window with a graphical tracer that shows the progress of the executing algorithm.

The reader is referred to Chapter 2 for more complete information on the GIDEN user interface.

³GIDEN is being developed under Windows9x/NT, Linux, and Solaris operating systems.

Core Environment Solver Interface

Core-GIDEN is an object-oriented environment (see [7] for a general description of object-oriented programming). Solvers are implemented as derived classes of a generic solver base class named `ExecBase` that is included in core-GIDEN. The constructor for the derived solver class is passed a `UserBase` object that provides several methods for communicating with the user interface; stable functions defined in the `UserBase` object are used by the solver class to report important changes (e.g., when items have moved between animation sets). Virtual methods in `ExecBase` provide for communication from core-GIDEN to the solver.⁴ In particular, through calls to these methods, core-GIDEN instructs the solver when to prepare for execution, when to execute, and when to terminate execution.

1.3.2 Solver Toolkit

Solvers contain the algorithm logic to solve network optimization problem instances. Solvers perform data manipulations, update the associated animation mechanisms, and inform core-GIDEN of the animation changes. As mentioned previously, solvers are implemented as derived classes of a generic solver base class named `ExecBase`. In this section we describe how GIDEN’s solver toolkit is organized; descriptions of the solvers included in the toolkit are given in Chapter 3.

While each network optimization problem has a variety of solution algorithms, the input and output of each algorithm for a given problem are typically the same. For this reason, algorithm implementations belonging to the solver toolkit are implemented as derived classes of an associated problem class, rather than as classes derived directly from `ExecBase`. For example, rather than having the solver for Dijkstra’s algorithm as a derived class of `ExecBase`, the solver class for Dijkstra’s algorithm is a derived class of the `ShortestPath` problem class which is derived directly from `ExecBase`. Figure 1.6 illustrates the relationship between individual solver classes and `ExecBase`. For a given problem all solvers will have a collection of common input and output requirements (e.g.,

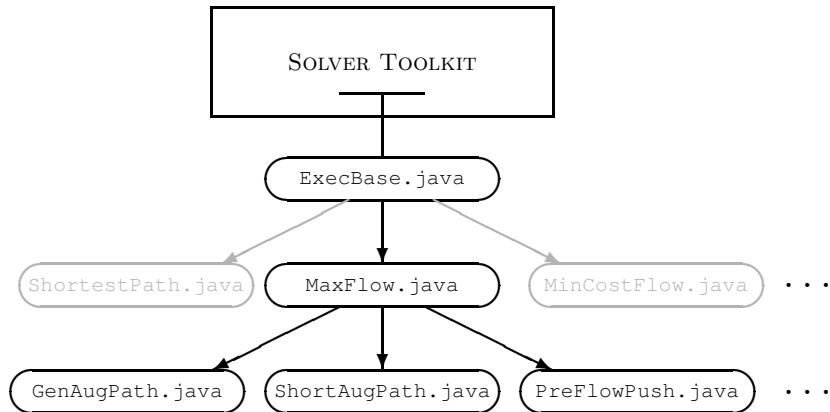


Figure 1.6: Class hierarchy for solver implementations.

all maximum-flow problem solvers require a source node, a sink node, and edge capacities as input, and provide edge flows and a minimum cut as output). Similarly, all solvers for a given problem will also have a collection of common animation sets, namely those associated with displaying problem instance solutions. The base class for each problem (e.g., `ShortestPath`) is designed to include all common characteristics for related solvers, allowing individual solver classes to benefit from code

⁴See Section 4.1.2 for a description of these methods.

reusability at the problem level. In addition to having the actual solver logic, individual solver classes are only required to have data checking routines and animation sets that differ from those of other solvers for the same problem.

Although the solver toolkit included with GIDEN covers a range of problems and is continually expanding, many users will want to add their own solvers to the toolkit. The following features of GIDEN facilitate solver development:

- prototypes for new solver implementations
- direct access to network data structures
- access to animation set creation and manipulation
- visual debugging through the user interface

The necessary tools for adding solvers to the solver toolkit are a text editor and the Java Development Kit (JDK), or some Java-based integrated development environment (IDE). Additionally, you must have access to the implementor's distribution of GIDEN.⁵ The process of developing new solvers and adding them to the toolkit is described in Chapter 4.

1.3.3 Network Data Structures

The network data structures library is based on standard graph data structures (see, for example, [9]). Node and edge arrays are provided to associate values with nodes and edges. Additional data structures include single and double linked lists, priority queues, queues, and other network related data structures. Both the single and double linked lists were developed from a common base class that supports all linked list methods. A common base that supports both types of linked lists provides a simple mechanism to switch between single and double linked lists without needing to edit or recompile the source code. Such libraries allow users to quickly build efficient network algorithm implementations. Our prototype version of GIDEN was implemented in the C++ programming language and used LEDA (Library of Efficient Data types and Algorithms) developed at the Max-Planck-Institut für Informatik (see [8]). The current data structures library used by GIDEN is based on a similar environment written by us in C++ (see [4]).

⁵Currently, only limited documentation is available to support GIDEN solver developers. For this reason, the implementor version of GIDEN is available only by special request. Please contact the authors for more information.

Chapter 2

Using the GIDEN Environment

2.1 The GIDEN User Interface

The GIDEN graphical user interface (GUI) features a single *controller window* and multiple *network windows*. In addition, each network window may have an auxiliary *pseudocode window* when it is operating in solver execution mode.

The main purpose of the GIDEN *controller window* (see Figure 2.1) is to create and manage all network windows, and to provide central access to the network editing tools. Additionally there are a few miscellaneous “global” operations that are available through the controller window.



Figure 2.1: GIDEN-2.0 controller window with “Edit Values” tool selected.

The second type of window is a *network window*. The controller window can create and manage multiple network windows simultaneously, each running in its own thread. Each network window can be used in two modes. In *edit mode*, a network window can be used to create and modify networks, as illustrated in Figure 2.2. A network window is in *solver mode* when executing an algorithm implementation from the GIDEN solver toolkit. An example of a network window in solver mode is shown in Figure 2.3, along with an auxiliary *pseudocode window*.

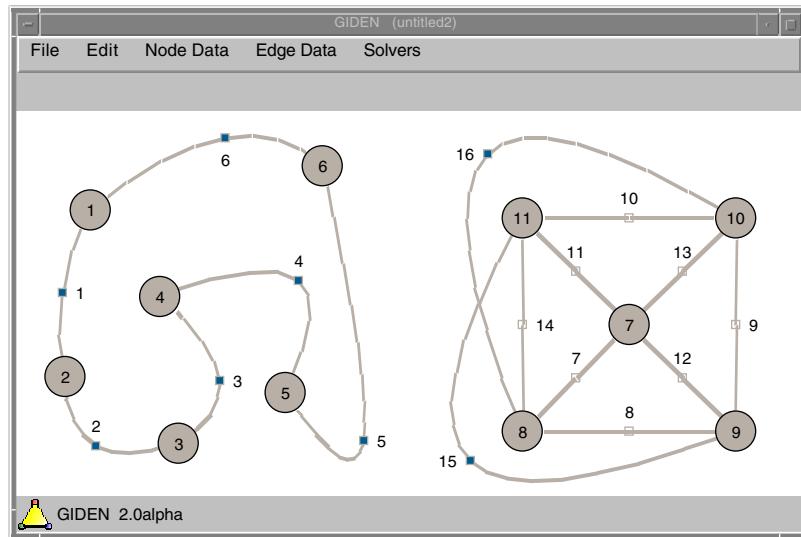


Figure 2.2: GIDEN-2.0 network window in edit mode.

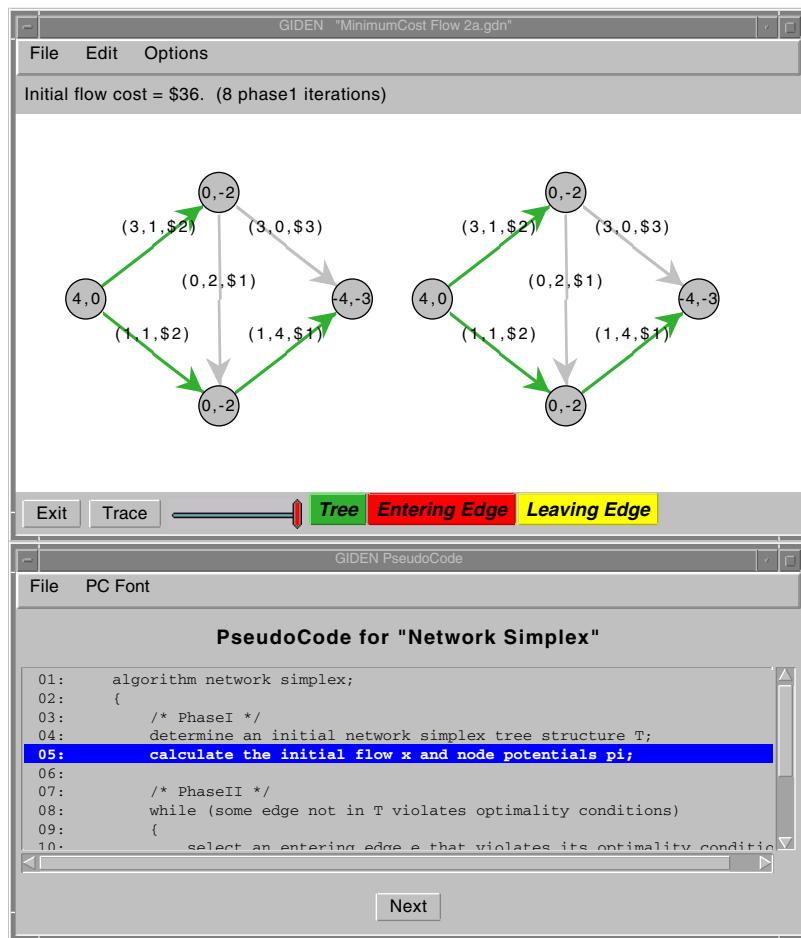


Figure 2.3: GIDEN-2.0 network window during solver execution with pseudocode displayed.

2.2 Building and Editing Networks

The GIDEN environment includes many features for building network instances. In this section we describe the facilities for drawing networks and for managing information associated with a network's nodes and edges. Throughout this section we assume that the network window of interest is in edit mode.

It is important to note that the current version of GIDEN *does not* include facilities to “undo” any editing action.

2.2.1 Drawing Networks

Network instances in GIDEN are created by placing nodes and edges on the “drawing canvas” of a network window that is in edit mode. There are several tools and operations available in GIDEN for drawing networks from scratch or for modifying existing networks. Many of the drawing operations use the “edit tools” that are available through the main controller window, while other operations and settings are accessible through the menu system of a network window.

Edit Tools

A prominent feature of the controller window is the collection of icons representing available “edit tools.” Each of these tools is used to control certain editing operations in network windows. An edit tool may be selected by clicking on the associated tool icon or by choosing the appropriate entry from the Tools menu in the controller window. Only one tool may be selected at any given time, and this selected tool is available to all network windows that are running in edit mode.

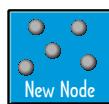
Most of the “edit tools” available in the controller window are used for drawing operations, such as creating, deleting, and placing nodes and edges on the canvas. Other tools are used to set the direction of edges, to control the placement of edge labels, and to edit information associated with nodes and edges. This section describes the purpose and usage of each of the edit tools.



The Trail Edge tool is used to draw a series of nodes and edges. After selecting the tool, the first click on the drawing canvas will either create a new node at the mouse position or, if a node already exists at that position, it will select that pre-existing node. Subsequent clicks on the canvas will create edges in series from the most recently selected node to the current mouse position, creating a new node at the current position if none exists.



The Star Edge tool is used to draw several edges emanating from a single node. After selecting the tool, the first click on the drawing canvas will either create a new node at the mouse position or, if a node already exists at that position, it will select that pre-existing node as the anchor node. Subsequent clicks on the canvas will create an edge between the anchor node and the current mouse position, creating a new node at the current position if none exists.



The New Node tool is used to create a new node on the drawing canvas. After selecting the tool, each click on the canvas will create a new node at the current mouse position. If a node already exists at the selected location, then no action is performed.



The New Edge tool is used to create a new edge on the drawing canvas. In order to create an edge $e = (i, j)$ between existing nodes i and j , first click on node i , and then click on node j .



The Reverse Edge tool is used to reverse the direction of an edge. When using this tool, selecting an edge $e = (i, j)$ will swap the head and tail nodes such that $e = (j, i)$. (*Note: This tool will still operate even when a network is being drawn as undirected. To see the direction of an edge, turn on the Directed Edges option in the network window's Edit menu.*)

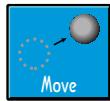


The Edit Values tool is used to edit information that is associated with the nodes and edges. To edit the currently displayed information, use the mouse to select the label of the target node or edge, and then type the new information in the resulting text input area. See Section 2.2.2 for more information on editing data fields.



The Select Item tool is used to select multiple nodes and edges that will either be deleted from or re-positioned on the drawing canvas. With this tool there are two methods for selecting nodes and edges. The first method is to use single mouse clicks to select individual nodes and edges. The second method is to “click and drag” the mouse over a region of the canvas, which results in selecting all items in the target region. Holding the `<shift>` key when using the first method will toggle the selection status of the item. Holding the `<shift>` key when specifying a region of the canvas will prevent previously-selected items from being automatically de-selected.

Once selected, nodes and edges can be repositioned by dragging them with the mouse while holding the `<control>` key, or by dragging with the middle mouse button on platforms that support a 3-button mouse. Fine tune positioning for selected items is available by using the Nudge Selected operations in the Edit menu. For more information on editing operations available for selected items, see Section 2.2.1 below.



The Move Item tool is used to re-position items on the drawing canvas. When this tool is selected, the cursor changes shape to indicate when the mouse is positioned over a movable item. Use the mouse to select and drag the target item from its current position to the desired location on the canvas. Items that can be moved using this tool include nodes, edge labels, and edge anchor points. For each of these items, details differ on the boundaries accepted for selecting an item, and the rules used to govern item positioning:

Selecting and Positioning Nodes. Nodes may be selected by pressing the mouse button anywhere in the boundary of the drawn node. The selected node can then be dragged to any position on the drawing canvas, without restriction.

Selecting and Positioning Edge Labels. The position of an edge label is tied to the location of the anchor point for the associated edge. An edge label is selected by pressing the mouse button anywhere in the boundary of the currently displayed label.¹ The edge label can then be positioned to the north (centered), south (centered), east (left aligned), or west (right aligned) of the edge anchor point.

¹If the edge data field currently displayed is empty for a particular edge, then the associated edge label cannot be re-positioned. In this case, temporarily change the displayed data to Edge ID (or some non-empty data field) before attempting to move the edge label.

Selecting and Positioning Edge Anchor Points. Edge anchor points are used to control the shape of the drawn edge and the general location of the associated edge label. An edge anchor point is selected by pressing the mouse button anywhere in the boundaries of the box region drawn at the current anchor point location. The selected edge anchor point can then be dragged to any position on the drawing canvas, without restriction. To allow edge anchor point to be automatically calculated, click the edge anchor point while holding the <shift> key.



The Delete Item tool is used to delete nodes and edges from the drawing canvas. To delete a node, click anywhere in the boundary of the drawn node. To delete an edge, click on the associated edge anchor point or edge label. Note that deleting a node results in all adjacent edges being deleted as well.

Edit Menu Operations and Settings

This section briefly describes the drawing operations and settings that are available through the Edit menu of a network window in edit mode. The scope of these menu selections is limited to the single target network window. Several of the menu choices perform operations that are available only when the Select Item tool is enabled; see the Select Item tool description above for more information.

Cut	Delete the selected nodes and edges from the drawing canvas. This choice is available only if the Select Item tool is enabled and there are nodes and/or edges that are currently selected.
Clear	Delete all nodes and edges on the drawing canvas. This choice is always available in edit mode.
Select All	Select all nodes and edges on the drawing canvas. This choice is available only if the Select Item tool is enabled.
Select All Nodes	Select all nodes on the drawing canvas. This choice is available only if the Select Item tool is enabled.
Nudge Selected	Move all selected nodes and edges by one pixel in the chosen direction. ² This choice is available only if the Select Item tool is enabled.
Enable Grid	Turn on a drawing grid for positioning nodes. Subsequent operations for adding or moving a node will align the node's center with a grid intersection point. ² This choice is always available in edit mode.
Directed Edges	This setting determines if an edge is drawn as directed or undirected. This choice is always available in edit mode, but may be overridden by an executing solver.

2.2.2 Managing Node and Edge Information

Associated with each network window in GIDEN is an underlying network object. This network object is composed of the following elements: a node set, an edge set, node data fields, and edge data fields. The member elements of the (possibly empty) node and edge sets correspond to the nodes and edges displayed on the network window's drawing canvas, and can thus be managed through the drawing operations described in Section 2.2.1. In this section we describe the facilities within GIDEN for managing the node and edge data fields.

The node and edge data fields (or “arrays”) are internally stored as `NodeArray` and `EdgeArray` objects (see [4]). Each node and edge array has a default *data type* associated with it, which

²The drawing grid is ignored when using the Nudge Selected operations to move nodes.

determines how the values of the array are parsed. Currently two data types are supported: “text” type and “integer” type. Values in a text array are represented using Java `String` objects, and values in an integer array are represented using Java `Integer` objects. Data fields are handled differently within GIDEN based on their associated type; for an example of this, see the topic “Editing Data Fields” in this section.

Default Data Fields

When a network object is constructed, it is created with three default data fields that are internally “owned” and managed by GIDEN. Because these fields are managed internally by the environment, the values associated with these fields cannot be edited directly by the user. These default fields are named `Node ID`, `Edge ID`, and `Euclidean Length`. The following text describes these fields and their intended usage:

<code>Node ID</code>	This is a node data field of text type. As the name suggests, the node array contains a unique identifier for each node in the network object. The primary purpose of the <code>Node ID</code> array is to distinguish nodes internally within GIDEN.
<code>Edge ID</code>	This is an edge data field of text type. Analogous to the <code>Node ID</code> array for nodes, this data field is used as a container for unique edge identifiers.
<code>Euclidean Length</code>	This is an edge data field of integer type. The array is used to store integer values that represent the Euclidean distance between an edge’s two endpoint nodes. Values are calculated using the center coordinates of the endpoint nodes. Note that the value associated with an edge gives the approximate length of that edge on the canvas if it is drawn as a straight line (e.g., if the edge’s anchor point is not manually positioned). The value associated with a given edge is automatically recalculated when either of its endpoint nodes is moved (e.g., with the <code>Move Item</code> tool). Although managed internally, the <code>Euclidean Length</code> array is provided as a convenience for end-users. The data array may be used as an input to any of the GIDEN solvers that require an integer type edge array as input (see Section 2.3.2).

User-Defined Data Fields

In addition to the default arrays that are associated with every network, users can create new node and edge data fields. These user-defined arrays are created through the `Add Data Field...` operation in a network window’s `Node Data` or `Edge Data` menu (see Section 2.2.2). Selecting this operation will evoke a dialog for specifying the following data field properties:

<code>Name</code>	This is a text identifier for the data field. This is the name that will be listed in the appropriate data field menu (see Section 2.2.2) and in the solver input dialogs (see Section 2.3.2). Valid names can include numbers, letters, spaces, and dashes. To avoid confusion, you cannot assign a name to a new node (edge) array that conflicts with an existing node (edge) array name. ³ Leading, trailing, and consecutive spaces will be removed from the name.
<code>Default Value</code>	This is the default value of an item in the array. When an array is created, this value is assigned to all existing nodes (edges). It will also be the value assigned to all newly created nodes (edges).

³For the purpose of comparison, names conflict if they are equal when ignoring case. For example, the name “length” conflicts with the name “Length”.

Type	This is the data type associated with the array. Currently two data types are supported: “text” type and “integer” type. Values in a text array are represented using Java <code>String</code> objects, and values in an integer array are represented using Java <code>Integer</code> objects. The underlying data type of an array determines the range of valid member values (as according to Java specifications).
------	---

The Data Field Menus

This section briefly describes the operations and settings that are available through the **Node Data** and **Edge Data** menus of a network window. Through these menus the user can create new arrays, delete existing arrays, and select the data fields that are displayed in the node and edge labels on the drawing canvas:

Creating New Arrays. New node and edge data fields are created using the **Add Data Field...** operation in the appropriate data menu. Selecting this operation will result in a dialog for specifying the properties of the new array. See the topic “User-Defined Data Fields” in this section for more information.

Deleting Existing Arrays. Existing data fields can be deleted from the network by selecting the array name through the **Delete Data Field** submenu. Only user-defined data fields may be deleted. In particular, the default data fields **Node ID**, **Edge ID**, and **Euclidean Length** are managed internally and can not be deleted (see Section 2.2.2).

Selecting Display Arrays. When a network window is in edit mode, each node and edge has an associated label that is displayed on the drawing canvas. To set which data field values are displayed in the node (edge) labels, select the appropriate array name from the top portion of the **Node Data** (**Edge Data**) menu.

Editing Data Fields

Node and edge data fields are edited using the **Edit Values** tool (see Section 2.2.1). Follow the steps below to edit a user-defined array:

1. Select the target array as the display array under the appropriate data menu in the network window.
2. Select the **Edit Values** tool from the controller window.
3. Use the mouse to select the label of the node or edge whose value you want to change. This should replace the label with a text input box that contains the current value.
4. Type the new data value in the text input box.
5. Press **<enter>** to accept the new value.

After entering a data value in the text input box, the new value is checked to ensure that it is valid for the underlying array data type. For example, integer type arrays will not accept values that are out of range for a Java `Integer`, or values that contain non-digit characters. If the specified value is invalid, then a warning will appear and the edit will be ignored.

2.3 Running Solvers

GIDEN users can create and modify networks using the mouse and keyboard as described in the previous section. Once a network is built (or opened), the user may select a solver implementation to apply to the displayed network. The process of running a GIDEN solver is outlined in the following steps:

1. Select the desired solver from the **Solvers** menu.
2. Specify the input data fields required by the solver (if any).
3. Execute the selected solver.
4. Interpret the solver results.

In this section we describe these steps more carefully.

2.3.1 Selecting a Solver

The first step in executing a solver is to select the solver name from the **Solvers** menu of the network window (only available in edit mode). The solvers currently available in GIDEN are categorized according to the underlying problem type. Currently, the solver toolkit provides solvers from four problem domains: minimum spanning tree problems, shortest path problems, maximum flow problems, and minimum-cost flow problems.⁴

After being chosen from the **Solvers** menu, the solver will analyze the current network object. If the solver is unable to cast the displayed network into an appropriate problem instance, then it returns immediately (leaving the network window in edit mode). Otherwise, the network window is switched to solver mode, and the user may be prompted for required input information, as described below.

2.3.2 Specifying Inputs

When a solver is selected from the toolkit, the target network window switches to solver mode. At this point, most GIDEN solvers will generate a dialog window to prompt the user to specify the node and edge input arrays required by the solver. This *solver input dialog* has two columns of information. The first column (left) lists the *logical names* of the needed solver inputs. These logical names are text identifiers that the solver uses to request and retrieve the input arrays. The second column (left) includes choice menus that list available candidate arrays (if any) that can be associated with the corresponding input logical name. The candidate arrays included in the choice menus are selected based on the kind of array (i.e., node or edge) and the underlying data type. (For example, the Prim solver will list all edge arrays of integer type as candidates for its “Length” input array.) After selecting the desired input arrays, click the **Accept** button. To exit the dialog without making selections, click the **Cancel** button. In this case, the solver will exit and the network window will return to edit mode. Note that some networks may not contain node or edge arrays that match the requirements of a given solver. In this case the dialog will indicate that the appropriate data is not available, and the **Accept** button will not be able to validate the input selections. To exit the solver, click the **Cancel** button.

Aside from the initial input dialog, some solvers will ask for user input during solver execution. In most cases, directions for specifying the needed input will appear in the status line of the network

⁴Some distributions also include heuristic solvers for the symmetric traveling salesman problem.

window. Typical examples would require the user to use the mouse to select a particular node, or use the keyboard to specify an option setting.

2.3.3 Controlling Execution

After selecting the needed input arrays (if any), the solver is ready to be run by the user. You will notice several features of a network window in solver mode that distinguish it from when it is running in edit mode. Most notably, the bottom area of the network window will contain several execution-related controls: an **Exit** button, an “action” button, an animation slider, and possibly several animation-set buttons. Each of these controls is described below:

- *The Exit Button:* Selecting this button will cause the solver to terminate at the earliest opportunity and will result in the network window returning to edit mode.
- *The Action Button:* The appearance and functionality of the action button depends on the state of the solver and several related settings. Depending on these settings, the action button will display one of the following labels: **Trace**, **Step**, **Go**, **Pause**, **Final**, or **Reset**.

When the action button is labeled **Reset**, the solver has completed execution. Selecting the action button when it displays this label causes the solver to begin execution again, as if the solver was selected from the **Solvers** menu. The topics “Final Animation Mode”, “Trace Animation Mode”, “Step Animation Mode”, and “Continuous Animation Mode” that appear later in this section describe the functionality of the action button when it displays other labels.

- *The Animation Slider:* The animation slider is used to determine the current animation mode (“final”, “trace”, or “continuous”). It is also used to control the speed of execution when operating in continuous animation mode. For more details, see the topics “Final Animation Mode”, “Trace Animation Mode”, and “Continuous Animation Mode” later in this section.
- *Animation-Set Buttons:* Animation-set buttons are used to specify which information about the current solver will be visually displayed during “trace” or “continuous” animation mode (see below). Each of the buttons corresponds to a solver-specific action or state, and visually represents that action or state using a pre-determined color. Under usual circumstances, each animation-set buttons can be either “enabled” or “disabled” to signify whether or not the underlying action or state is to be displayed. The state of an animation-set button is toggled by clicking on the button when in “trace” mode or “continuous” mode. For details on the use of animation-set buttons, see the topic “Trace Animation Mode” later in this section. See Chapter 3 for information on the animation sets provided with each of the GIDEN solvers.

There are several different ways to execute a GIDEN solver, depending on the level of visualization that is desired. We will describe the basics here, but the best way to learn about the algorithm animation capabilities within GIDEN is to experiment with the various solvers and settings. We will cast our discussion of solver execution by describing five different “modes” of animation: final animation mode, trace animation mode, step animation mode, continuous animation mode, and pseudocode animation mode.

Final Animation Mode

Final animation mode is used to execute a given solver without any animation. To run in final mode, position the animation slider to the far left (so that the action button is named **Final**), and select the action button. This will execute the solver until it terminates — usually with an optimal solution or with a certificate of infeasibility. At conclusion of solver execution, the action button name will change to **Reset**.

Trace Animation Mode

Trace animation is intended as the default animation mode. When running a solver in trace mode, the detail of solver animation depends on the state of the animation-set buttons. In particular, the solver will suspend execution at certain epochs that correspond to the action or underlying state of an enabled animation-set button. Each time execution is suspended, the network is redrawn to reflect the current state of the solver.

To run a solver in trace mode, first position the animation slider to the far right, and make sure that the **Detail Animation** setting is turned off in the **Options** menu. The action button should now display the name **Trace**. To execute the solver, use the mouse to repeatedly select the **Trace** button.

You can think of trace animation mode as follows: Each click on the **Trace** button says to the solver “*start running until something interesting happens*”, where the solver decides if something “interesting” happened based on which animation-set buttons are enabled.

Step Animation Mode

Note: Step animation mode is intended to be used primarily for solver debugging purposes. We recommend against end users executing solvers in step mode.

Step mode is similar to trace mode with two major differences: (1) All animation-set buttons are treated as enabled, and (2) solver execution suspends whenever any item is placed in an animation set. Running a solver in step mode is similar to trace mode, except that the **Detail Animation** setting must be turned on in the **Options** menu.

Continuous Animation Mode

When executing in trace mode, a solver will suspend execution at certain epochs that correspond to “interesting” events. Then to resume solver execution, the user must manually select the **Trace** button. Continuous animation mode is similar to trace mode except that there is a built-in time delay between when a solver suspends, and when it resumes execution. The amount of this delay is determined by the position of the animation slider.

To execute a solver in continuous animation mode, first position the animation slider strictly between the far left and the far right positions. (The action button name changes to **Go** when the slider is positioned between the two extremes.) The built-in delay will decrease as the slider is moved to the left and increase as it is moved to the right.⁵ After enabling the animation-set buttons of interest, press the **Go** button. This will start execution and change the action button name to **Pause**. The solver will continue to execute until it terminates or until you select the **Pause** button.

Continuous animation mode can be used with the **Detail Animation** setting either enabled or disabled. When detail animation is enabled, the behavior of continuous animation is similar to executing in step mode with a built-in delay. If the setting is not enabled (as by default), then continuous animation is similar to trace mode with an automatic delay.

⁵It may help to think of “final mode” as the far left extreme, where there is no delay, and of “trace mode” as the far right extreme, where the delay is infinite. However the continuous animation delay is actually a linear function between two finite values; the maximum automatic delay is approximately 5 seconds.

Pseudocode Animation Mode

As shown in Figure 2.3, certain GIDEN solvers will display an auxiliary window with a text-based pseudocode description of the algorithm. This *pseudocode window* contains a **Next** button, that is used to control solver execution by stepping through the algorithm logic. Each time the user selects the **Next** button, the solver resumes execution until it encounters the next algorithmic step that is represented in the pseudocode. The solver then suspends execution, highlights the current line of pseudocode, and redraws the network to reflect the state of the solver. Using this approach to control solver execution is what we mean by running in pseudocode mode.

Trace, step, and continuous animation modes all depend on the animation sets that are present (and enabled) in a solver. In contrast, the pseudocode animation mode is entirely independent of the available animation sets, and instead is driven by the text description of the underlying algorithm.

To execute a solver in pseudocode mode, the auxiliary pseudocode window must be visible. If this window is not visible by default (it will depend on the position of the network window and other factors), you may evoke the window by enabling the **Show Pseudo-Code Window** setting from the Options menu. Then click repeatedly on the **Next** button to step through solver execution.

Please note that not all solvers provide a pseudocode description of the algorithm. For solvers that do not support pseudocode animation mode, the **Show Pseudo-Code Window** setting will be automatically disabled.

2.3.4 Interpreting Results

Once a solver has found an optimal solution,⁶ or determined that no such solution exists, it will terminate execution. Upon termination, most GIDEN solvers will display information about the final solution (possibly including a “certificate” of optimality), or they will provide a certificate of infeasibility. In either case, usually the animation-set buttons will be replaced with a “result key” that can be used to interpret the solver’s final result. Additional information about the solution may be provided in the network window’s status line. The user is referred to Chapter 3 for information on the particular result information provided by each of the default GIDEN solvers.

⁶Some solvers may not yield an optimal solution, but may yield a “good” solution by some measure. These solvers are known as “heuristics.”

Chapter 3

Solver Reference

In this chapter, we describe the network optimization problems that can be solved using the GIDEN solver toolkit, and we give information on available solvers. The information we present for each solver includes algorithm pseudocode that closely follows the algorithm descriptions presented in the text *Network Flows: Theory, Algorithms, and Application*, by Ahuja, Magnanit, and Orlin [1]; we have made it a priority to include the pseudocode exactly as it appears in [1] whenever possible. The following assumptions are made for all solvers in the GIDEN solver toolkit:

1. The network does not contain any parallel edges.
2. The network does not contain any loops.
3. Lower bounds on edge flows are zero-valued.

At present, GIDEN's solver toolkit includes implementations for the spanning tree, shortest path, maximum flow, and minimum-cost flow problems:

Minimum Spanning Tree

- Kruskal's algorithm
- Prim's algorithm

Shortest Path

- Dijkstra's algorithm
- FIFO label correcting algorithm

Maximum Flow

- Generic augmenting path algorithm
- Shortest augmenting path algorithm
- Preflow-push algorithm

Minimum-Cost Flow

- Capacity scaling algorithm
- Cycle canceling algorithm
- Out-of-kilter algorithm

- Network simplex algorithm
- Successive shortest path algorithm

Additional solver implementations are under development and testing.

3.1 Minimum Spanning Tree Problem

Given an undirected network $\mathcal{N} = (G, l)$, the *minimum spanning tree problem* is to find a spanning tree of G with minimal length, where the length of a tree T is calculated as $\sum_{e \in T} l_e$.

Input: An undirected network $\mathcal{N} = (G, l)$, where $l \in \mathbb{R}^A$.

Formulation:

$$\begin{aligned} \min \quad & \sum_{e \in T} l_e \\ \text{s.t.} \quad & T \text{ is a spanning tree of } G. \end{aligned}$$

Feasible Output: A minimum spanning tree T^* and the tree's total length $\sum_{e \in T^*} l_e$.

Infeasible Output: A proper subset R of N , such that no edges (i, j) exist with $i \in R$ and $j \in N \setminus R$. We also give either a forest F^* composed of a minimum spanning tree T^* for each component of G , or a minimum spanning tree for the component of G with node set R .

Required Problem Input

GIDEN solvers derived from the MinSpan problem base class will request the following data field in the input dialog:

“Length” — edge array of integer data type;
provides edge lengths l_e for each $e \in A$.

3.1.1 Kruskal's Algorithm

```

algorithm kruskal;
{
    sort edges in nondecreasing order of length;
    LIST := ∅;
    while (|LIST| < |N| - 1 and unexamined edges exist) do
    {
        e := unexamined edge with minimal length;
        if (adding e to LIST does not create a cycle) then
            add e to LIST;
        else
            discard e;
    }
} (END of 'kruskal')

```

Solver-Specific Input: (none)

Solver Animation Sets:

Trial (red): An edge being considered for inclusion on LIST.

Acquired (green): Edges included on LIST and nodes with an incident edge on LIST.

Discarded (yellow): Edges $e = (i, j)$ that were considered for inclusion in LIST but were rejected because adding e to LIST would have created a cycle in the component of G containing node i and node j .

Label Information:

Edge labels: Edge length l_e for each edge $e \in A$.

Node labels: None.

Feasible Result Information:

Minimum Spanning Tree (orange): Used to color the edges in a minimum spanning tree of G . The tree length is reported in the network window status line.

Infeasible Result Information:

Forest of Component-wise Minimum Spanning Trees (orange): A minimum spanning tree is found for each component; this is used to color the edges in each such spanning tree. The total forest length is reported in the network window status line.

3.1.2 Prim's Algorithm

```

algorithm prim;
{
    /* initialize algorithm information */
     $d_i := \infty$  and  $\text{pred}(i) := \text{null}$  for all  $i \in N$ ;
     $P := \emptyset$ ,  $L := \emptyset$ ,  $U := N$ ;

    /* setup for starting at node  $s$  */
     $d_s := 0$ ;
    move node  $s$  from set  $U$  to set  $L$ ;

    while (the set  $L$  is non-empty) do
    {
        select node  $i \in L$  such that  $d_i \leq d_j$  for all  $j \in L$ ;
        move node  $i$  from set  $L$  to set  $P$ ;

        for (all nodes  $j$  with  $(i,j) \in A$ ) do
        {
            if ( $d_j > l_{(i,j)}$ ) then
            {
                 $d_j := l_{(i,j)}$ ;
                 $\text{pred}(j) := i$ ;
                if ( $j \in U$ ) then move  $j$  from set  $U$  to set  $L$ ;
            }
        }
    }
} (END of 'prim')

```

Solver-Specific Input:

s — start node for searching the network

Solver Animation Sets:

Trial (red): Nodes $i \in L$ and edges $(i, \text{pred}(i))$ for nodes $i \in L$ such that $\text{pred}(i) \neq \text{null}$.

Acquired (green): Nodes $i \in P$ and edges $(i, \text{pred}(i))$ for nodes $i \in P$ such that $\text{pred}(i) \neq \text{null}$.

Discarded (yellow): An edge $(i,j) \in A$ such that $i \in P$ and $j \in L$ (or vice versa) is “discarded” if it is considered as a candidate predecessor edge for node j but is rejected because either

- (i) at the time (i,j) is considered, node j already has a predecessor edge of length no greater than the length of (i,j) , i.e., $l_{(\text{pred}(j),j)} \leq l_{(i,j)}$, or
- (ii) a predecessor edge of shorter length is later found, before node j is permanently labeled.

Label Information:

Edge labels: Edge length l_e for each edge $e \in A$.

Node labels: The length $d_i \equiv l_{(\text{pred}(i),i)}$ of the predecessor edge $(\text{pred}(i),i)$ from each node $i \in N$ to the connected component R . Nodes with $d_i = \infty$ are labeled “-”.

Feasible Result Information:

Minimum Spanning Tree (orange): Used to color the edges in a minimum spanning tree of G . The tree length is reported in the network window status line.

Infeasible Result Information:

Minimum Spanning Tree on R (orange): A minimum spanning tree for the component R of the network that contains the specified starting node s . The length of the minimum spanning tree on component R is reported in the network window status line.

3.2 Shortest Path Problem

Given a directed network $\mathcal{N} = (G, l, s)$, the *shortest path problem* is to find a set of dipaths from node s to each node $i \in N \setminus \{s\}$ with minimal length, where the length of an s - i dipath P_{si} for $i \in N \setminus \{s\}$ is calculated as $\sum_{e \in P_{si}} l_e$.

Input: A directed network $\mathcal{N} = (G, l, s)$, where $l \in \mathbb{R}^A$ and $s \in N$.

Formulation:

For each node $i \in N \setminus \{s\}$, $\min \sum_{e \in P_{si}} l_e$ s.t. P_{si} is an s - i dipath in G .

Feasible Output: A shortest path tree T^* that includes a shortest path P_{si}^* for each $i \in N \setminus \{s\}$, the tree's total length $\sum_{e \in T^*} l_e$, the shortest path lengths to each node $i \in N$, and the sum of the shortest path lengths, $\sum_{i \in N} \sum_{e \in P_{s,i}^*} l_e$.

Infeasible Output: We return one of the following: (1) A proper subset R of N , such that $s \in R$ and there are no edges $(i, j) \in A$ such that $i \in R$ and $j \notin R$; in this case, we also give the shortest path tree from s to all nodes in R . (2) A negative length cycle C in the component of G containing s ; in this case we also give the length of the cycle, $\sum_{e \in C} l_e < 0$.

Required Problem Input

GIDEN solvers derived from the `ShortestPath` problem base class will request the following information:

- “Length” — edge array of integer data type;
provides edge lengths l_e for each $e \in A$.
- s — start node for searching the network

The “Length” data field is requested in the solver input dialog, and the source node s is requested at the start of solver execution.

3.2.1 Dijkstra's Algorithm

```

algorithm dijkstra;
{
    /* initialize algorithm information */
     $d_i := \infty$  and  $\text{pred}(i) := \text{null}$  for all  $i \in N$ ;
     $P := \emptyset$ ,  $L := \emptyset$ ,  $U := N$ ;

    /* setup for starting at node  $s$  */
     $d_s := 0$ ;
    move node  $s$  from set  $U$  to set  $L$ ;

    while (the set  $L$  is non-empty) do
    {
        select node  $i \in L$  such that  $d_i \leq d_j$  for all  $j \in L$ ;
        move node  $i$  from set  $L$  to set  $P$ ;

        for (all nodes  $j$  with  $(i, j) \in A$ ) do
        {
            if ( $d_j > d_i + l_{(i,j)}$ ) then
            {
                 $d_j := d_i + l_{(i,j)}$ ;
                 $\text{pred}(j) := i$ ;
                if ( $j \in U$ ) then move  $j$  from set  $U$  to set  $L$ ;
            }
        }
    }
} (END of 'dijkstra')

```

Solver-Specific Input:

Additional requirement that the selected “Length” input array can contain only non-negative values.

Solver Animation Sets:

Trial (red): Nodes $i \in L$ and edges $(i, \text{pred}(i))$ for nodes $i \in L$ such that $\text{pred}(i) \neq \text{null}$.

Acquired (green): Nodes $i \in P$ and edges $(i, \text{pred}(i))$ for nodes $i \in P$ such that $\text{pred}(i) \neq \text{null}$.

Discarded (yellow): An edge $(i, j) \in A$ such that $i \in P$ and $j \in L$ is “discarded” if it is considered as a candidate predecessor edge for node j but is either

- (i) rejected immediately, because an $s - j$ dipath has already been found that is at least as short as the $s - j$ dipath through (i, j) , i.e., $d_j \leq d_i + l_{(i,j)}$, or
- (ii) rejected later, because a shorter $s - j$ dipath is found before node j is permanently labeled.

Label Information:

Edge labels: Edge length l_e for each edge $e \in A$.

Node labels: Current “node distance” d_i from the starting node s . Nodes with $d_i = \infty$ are labeled “-”. Node s is labeled “s” initially and then with its “distance label,” $d_s = 0$.

Feasible Result Information:

Shortest Path Tree (orange): Used to color a shortest path tree rooted at s and spanning the nodes of G . The total tree length, along with the sum of (finite) path lengths is reported in the network window status line. The node labels report the final distances d_i for each $i \in N$.

Infeasible Result Information:

Shortest Path Tree (orange): Used to color a shortest path tree rooted at s for the component R of nodes and edges that are di-path reachable from s . This tree length, along with the sum of (finite) path lengths is reported in the network window status line.

3.2.2 FIFO Label Correcting Algorithm

```

algorithm fifo_label_correcting;a
{
    ds := 0 and pred(s) := null;
    dj :=  $\infty$  for each node j  $\in N \setminus \{s\}$ ;
    QUEUE := {s};
    while (QUEUE not empty) do
    {
        remove a node i from QUEUE;
        for (each edge  $(i, j) \in A(i)$ ) do
        {
            if (dj > di +  $l_{(i,j)}$ ) then
            {
                dj := di +  $l_{(i,j)}$ ;
                pred(j) := i;
                if (j  $\notin$  QUEUE) then add j to QUEUE;
            }
        }
    }
}
} (END of 'fifo_label_correcting')

```

^aSee Chapter 5, Sections 3-5 of [1] for a description of the FIFO label correcting algorithm as presented here.

Solver-Specific Input: (none)

Solver Animation Sets:

Accepted (green): Nodes $i \in N$ such that $d_i < \infty$, and edges $(\text{pred}(i), i)$ for each i such that $i \neq s$ (whose predecessor node is undefined).

Current (red): The node i taken from the QUEUE whose outgoing edges are being examined, and the edge that is currently being examined.

Discarded (yellow): An edge $(i, j) \in A$ such that $i \in P$ and $j \in L$ (or vice versa) is “discarded” if it is considered as a candidate predecessor edge for node j but is rejected because either

- (i) node j already has a predecessor edge of length no greater than the length of (i, j) , i.e., $l_{(\text{pred}(j), j)} \leq l_{(i,j)}$, or
- (ii) a predecessor edge of shorter length is later found before node j is permanently labeled.

Label Information:

Edge labels: Edge length l_e for each edge $e \in A$.

Node labels: Current “node distance” from the acquired component, d_i . Nodes with $d_i = \infty$ are labeled “-”. Node s is labeled “s”; its “distance” is equal to zero.

Feasible Result Information:

Shortest Path Tree (orange): Used to color a shortest path tree rooted at s and spanning the nodes of G . The total tree length, along with the sum of (finite) path lengths is reported in the network window status line. The node labels report the final distances d_i for each $i \in N$.

Infeasible Result Information:

Shortest Path Tree (orange): Used to color a shortest path tree rooted at s for the component R of nodes and edges that are di-path reachable from s . This tree length, along with the sum of (finite) path lengths is reported in the network window status line.

Negative-Cost Cycle (blue): Used to color a directed cycle in the network with negative total length (this is known as a “negative cost cycle”). The length (cost) of the cycle is reported in the network window status line.

Note that the infeasible result information that is reported depends on the component R of nodes and edges that are di-path reachable from s . If this component contains a negative cost cycle, then this cycle will be reported. Otherwise, only the shortest path tree on component R will be reported — even if there is a negative cost cycle present elsewhere in the network.

3.3 Maximum Flow Problem

Given a directed network $\mathcal{N} = (G, u, s, t)$, the *maximum flow problem* is to find a feasible s - t flow of maximal value, where the value of a feasible flow x is calculated as $\sum_{\{i:(i,t) \in A\}} x_{(i,t)}$. We say a flow x is *feasible* for the maximum flow problem if it satisfies the bound constraints, $0 \leq x \leq u$, and the balance constraints:

$$\sum_{\{j:(i,j) \in A\}} x_{(i,j)} = \sum_{\{j:(j,i) \in A\}} x_{(j,i)} \quad \text{for each node } i \in N \setminus \{s, t\}.$$

Input: A network $\mathcal{N} = (G, u, s, t)$, where $u \in \mathbb{Z}_+^A$ and $s, t \in N$.

Formulation:

max	$\sum_{\{i:(i,t) \in A\}} x_{(i,t)}$	
s.t.	$\sum_{\{j:(i,j) \in A\}} x_{(i,j)} = \sum_{\{j:(j,i) \in A\}} x_{(j,i)}$	for each node $i \in N \setminus \{s, t\}$.
	$0 \leq x \leq u$	

Feasible Output: A maximum flow x^* , the maximum flow value $\sum_{\{i:(i,t) \in A\}} x_{(i,t)}^*$, and an s - t cut $\delta(S, \bar{S})$ such that $s \in S$, $t \in \bar{S}$ and $\sum_{\{(i,j) \in A : i \in S, j \in \bar{S}\}} u_{(i,j)} = \sum_{\{i:(i,t) \in A\}} x_{(i,t)}^*$.

Infeasible Output: *The maximum-flow problem, as we have described it, is never infeasible since the flow $x = 0$ satisfies the bound constraints and the balance constraints for all problem instances.*

Required Problem Input

GIDEN solvers derived from the MaxFlow problem base class will request the following information:

- “Capacity” — edge array of integer data type;
provides edge capacities u_e for each $e \in A$.
- s — source node
- t — sink node

The “Capacity” data field is requested in the solver input dialog. Source node s and sink node t are requested at the start of solver execution.

3.3.1 Generic Augmenting Path Algorithm

```

algorithm generic augmenting path;a
{
     $x := 0;$ 
    while ( $G^x$  contains an  $s-t$  dipath) do
    {
        identify an  $s-t$  dipath  $P$  in  $G^x$ ;
        calculate  $\delta := \min_{\{(i,j) \in P\}} u_{(i,j)}^x$ ;
        augment  $\delta$  units of flow along  $P$  and update  $G^x$ ;
    }
} (END of 'generic augmenting path')

```

^aSee Chapter 6, Section 4 of [1] for a description of the generic augmenting path algorithm as presented here.

Solver-Specific Input:

The user is asked to specify a search method for finding an augmenting path. The current implementation supports only two choices: “breath-first search” (BFS) and “depth-first search” (DFS). This input is requested through the network window status line.

Solver Animation Sets:

Trial (red): Nodes and edges being considered for inclusion in an augmenting path of G .

Acquired (green): Nodes and edges in the current augmenting path of G .

Discarded (yellow): Edges that are encountered during the search for an augmenting path but are disregarded because their ends have both already been “seen” during the course of the search, or because no residual capacity is available.

Label Information:

Edge labels: Labels on the edges $e \in A$ give the current flow and the remaining capacity: $(x_e, u_e - x_e)$

Node labels: Source node s is labeled “s”, and sink node t is labeled “t”; other nodes have empty labels.

Feasible Result Information:

Reachable Nodes (orange): Used to color the set R of nodes that are reachable from s in G^x at the end of solver execution.

Minimum Capacity Cut (black): Used to color the edges in the minimum cut $\delta(R, N \setminus R)$ at the end of solver execution.

The final flow value and the minimum cut capacity are displayed in the status line.

3.3.2 Shortest Augmenting Path Algorithm

```

algorithm shortest augmenting path;a
{
     $x := 0$ ;
    obtain the exact distance labels  $d_i$  for each node  $i \in N$ ;b
     $i := s$ ;
    while ( $d_s < |N|$ ) do
    {
        if ( $i$  has an admissible edgec) then
        {
            advance( $i$ );
            if ( $i = t$ ) then
            {
                augment;
                 $i := s$ ;
            }
            else retreat( $i$ );
        }
    } (END of 'shortest augmenting path')

procedure advance( $i$ );
{
    let  $(i, j)$  be an admissible edge in  $A^x(i)$ ;
    pred( $j$ ) :=  $i$  and  $i := j$ ;
}

procedure retreat( $i$ );
{
     $d_i := \min_{\{(i,j) \in A^x(i) : u_{(i,j)}^x > 0\}} d_j + 1$ ;
    if ( $i \neq s$ ) then  $i := \text{pred}(i)$ ;
}

procedure augment;
{
    using the pred indices, identify the augmenting  $s-t$  path  $P$ ;
     $\delta := \min_{\{(i,j) \in P\}} u_{(i,j)}^x$ ;
    augment  $\delta$  units of flow along path  $P$ ;
}

```

^aSee Chapter 7, Section 4 of [1] for a description of the shortest augmenting path algorithm as presented here.

^bFor each node $i \in N \setminus \{t\}$, $d_i :=$ the minimum number of edges in an $i-t$ dipath in G .

^cAn edge (i, j) in G^x is *admissible* if $d_j = d_i + 1$ and $u_{(i,j)}^x > 0$.

Solver-Specific Input:

There is a solver option to use “smart label updates”; this option, if selected, incorporates into the solver an improved approach for updating distance labels as suggested on pp. 219–220 in [1].

Solver Animation Sets:

Trial (red): Nodes and edges being considered for inclusion in an augmenting s - t dipath of G^x .

Acquired (green): Nodes and edges in the current augmenting s - t dipath of G^x .

Discarded (yellow): Inadmissible edges that are encountered during the search for an augmenting s - t dipath. Nodes whose distance labels have changed during the current search for an augmenting s - t dipath.

Label Information:

Edge labels: Labels on the edges $e \in A$ give the current flow and the remaining capacity:
 $(x_e, u_e - x_e)$

Node labels: Current distance labels d_i for nodes $i \in N \setminus \{t\}$. Sink node t is labeled “t”
 $(d_t = 0)$.

Feasible Result Information:

Reachable Nodes (orange): Used to color the set R of nodes that are reachable from s in G^x at the end of solver execution. set

Minimum Capacity Cut (black): Used to color the edges in the minimum cut $\delta(R, N \setminus R)$ at the end of solver execution.

The final flow value and the minimum cut capacity are displayed in the status line.

Bibliography

- [1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [2] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier North-Holland, 1976.
- [3] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, 1998.
- [4] David Dilworth, Collette Couillard, and Jonathan H. Owen. Graph & related data structures user's guide. Technical Report TR-96.09, Department of Industrial Engineering and Management Sciences, Northwestern University, 1996.
- [5] Christopher V. Jones. *Visualization and Optimization*. Operations Research / Computer Science Interfaces Series. Kluwer Academic Publishers, Boston, 1996.
- [6] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Saunders College Publishing, Fort Worth, 1976.
- [7] Bertrand Meyer. *Object Oriented Software Construction*. Prentice Hall, second edition, 1996.
- [8] Stefan Näher and Christian Uhrig. *The LEDA User Manual, Version R-3.3*, 1996. Available via anonymous ftp from [ftp.mpi-sb.mpg.de](ftp://ftp.mpi-sb.mpg.de) in /pub/LEDA.
- [9] Robert Endre Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NFS Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.

Game Theory

In previous chapters, we have encountered many situations in which a *single* decision maker chooses an optimal decision without reference to the effect that the decision has on other decision makers (and without reference to the effect that the decisions of others have on him or her). In many business situations, however, two or more decision makers simultaneously choose an action, and the action chosen by each player affects the rewards earned by the other players. For example, each fast food company must determine an advertising and pricing policy for its product, and each company's decision will affect the revenues and profits of other fast food companies.

Game theory is useful for making decisions in cases where two or more decision makers have conflicting interests. Most of our study of game theory deals with situations where there are only two decision makers (or players), but we briefly study *n*-person (where $n > 2$) game theory also. We begin our study of game theory with a discussion of two-player games in which the players have no common interest.

15.1 Two-Person Zero-Sum and Constant-Sum Games: Saddle Points

Characteristics of Two-Person Zero-Sum Games

- 1 There are two players (called the row player and column player).
- 2 The row player must choose 1 of m strategies. Simultaneously, the column player must choose 1 of n strategies.
- 3 If the row player chooses her i th strategy and the column player chooses her j th strategy, the row player receives a reward of a_{ij} and the column player loses an amount a_{ij} . Thus, we may think of the row player's reward of a_{ij} as coming from the column player.

Such a game is called a **two-person zero-sum game**, which is represented by the matrix in Table 1 (the game's **reward matrix**). As previously stated, a_{ij} is the row player's reward

TABLE 1
Example of Two-Person Zero-Sum Game

Row Player's Strategy	Column Player's Strategy			
	Column 1	Column 2	...	Column n
Row 1	a_{11}	a_{12}	...	a_{1n}
Row 2	a_{21}	a_{22}	...	a_{2n}
⋮	⋮	⋮		⋮
Row m	a_{m1}	a_{m2}	...	a_{mn}

(and the column player's loss) if the row player chooses her i th strategy and the column player chooses her j th column strategy.

For example, in the two-person zero-sum game in Table 2, the row player would receive two units (and the column player would lose two units) if the row player chose her second strategy and the column player chose her first strategy.

A two-person zero-sum game has the property that for any choice of strategies, the sum of the rewards to the players is zero. In a zero-sum game, every dollar that one player wins comes out of the other player's pocket, so the two players have totally conflicting interests. Thus, cooperation between the two players would not occur.

John von Neumann and Oskar Morgenstern developed a theory of how two-person zero-sum games should be played, based on the following assumption.

Basic Assumption of Two-Person Zero-Sum Game Theory

Each player chooses a strategy that enables him to do the best he can, given that his opponent *knows the strategy he is following*. Let's use this assumption to determine how the row and column players should play the two-person zero-sum game in Table 3.

How should the row player play this game? If he chooses row 1, the assumption implies that the column player will choose column 1 or column 2 and hold the row player to a reward of four units (the smallest number in row 1 of the game matrix). Similarly, if the row player chooses row 2, the assumption implies that the column player will choose column 3 and hold the row player's reward to one unit (the smallest or minimum number in the second row of the game matrix). If the row player chooses row 3, he will be held to the smallest number in the third row (5). Thus, the assumption implies that the row player should choose the row having the largest minimum. Since $\max(4, 1, 5) = 5$, the row player should choose row 3. By choosing row 3, the row player can ensure that he will win at least $\max(\text{row minimum}) = \text{five units}$.

From the column player's viewpoint, if he chooses column 1, the row player will choose the strategy that makes the column player's losses as large as possible (and the row player's winnings as large as possible). Thus, if the column player chooses column 1, the row player will choose row 3 (because the largest number in the first column is the 6 in the third row). Similarly, if the column player chooses column 2, the row player will again choose row 3, because $5 = \max(4, 3, 5)$. Finally, if the column player chooses column 3, the row player will choose row 1, causing the column player to lose $10 = \max(10, 1, 7)$ units. Thus,

TABLE 2

1	2	3	-1
2	1	-2	0

TABLE 3
A Game with a Saddle Point

Row Player's Strategy	Column Player's Strategy			Row Minimum
	Column 1	Column 2	Column 3	
Row 1	4	4	10	4
Row 2	2	3	1	1
Row 3	6	5	7	5
Column Maximum	6	5	10	

the column player can hold his losses to $\min(\text{column maximum}) = \min(6, 5, 10) = 5$ by choosing column 2.

We have shown that the row player can ensure that he will win at least five units and the column player can hold the row player's winnings to at most five units. Thus, the only rational outcome of this game is for the row player to win exactly five units; the row player cannot expect to win more than five units, because the column player (by choosing column 2) can hold the row player's winnings to five units.

The game matrix we have just analyzed has the property of satisfying the **saddle point condition**:

$$\max_{\text{all rows}} (\text{row minimum}) = \min_{\text{all columns}} (\text{column maximum}) \quad (1)$$

Any two-person zero-sum game satisfying (1) is said to have a **saddle point**. If a two-person zero-sum game has a saddle point, the row player should choose any strategy (row) attaining the maximum on the left side of (1). The column player should choose any strategy (column) attaining the minimum on the right side of (1). Thus, for the game we have just analyzed, a saddle point occurred where the row player chose row 3 and the column player chose column 2. The row player could make sure of receiving a reward of at least five units (by choosing the optimal strategy of row 3), and the column player could ensure that the row player would receive a reward of at most five units (by choosing the optimal strategy of column 2). If a game has a saddle point, we call the common value of both sides of (1) the **value (v)** of the game to the row player. Thus, this game has a value of 5.

An easy way to spot a saddle point is to observe that the reward for a saddle point must be the smallest number in its row and the largest number in its column (see Problem 4 at the end of this section). Thus, like the center point of a horse's saddle, a saddle point for a two-person zero-sum game is a local minimum in one direction (looking across the row) and a local maximum in another direction (looking up and down the column).

A saddle point can also be thought of as an **equilibrium point** in that neither player can benefit from a unilateral change in strategy. For example, if the row player were to change from the optimal strategy of row 3 (to either row 1 or row 2), his reward would decrease, while if the column player changed from his optimal strategy of column 2 (to either column 1 or column 3), the row player's reward (and the column player's losses) would increase. Thus, a saddle point is stable in that neither player has an incentive to move away from it.

Many two-person zero-sum games do not have saddle points. For example, the game in Table 4 does not have a saddle point, because

$$\max(\text{row minimum}) = -1 < \min(\text{column maximum}) = +1$$

In Sections 15.2 and 15.3, we explain how to find the value and the optimal strategies for two-person zero-sum games that do not have saddle points.

TABLE 4
A Game with No Saddle Point

Row Player's Strategy	Column Player's Strategy		Row Minimum
	Column 1	Column 2	
Row 1	-1	+1	-1
Row 2	+1	-1	-1
Column Maximum	+1	+1	

$0) = 5$ by
units and
, the only
row player
g column

saddle point

(I)

If a two-
gy (row)
strategy
have just
in player
ive units
that the
strategy
es of (I)

int must
lem 4 at
oint for
the row)

ayer can
change
crease,
column
increase.
from it.
game in

gies for

Two-Person Constant-Sum Games

Even if a two-person game is not zero-sum, two players can still be in total conflict. To illustrate this, we now consider two-person constant-sum games.

DEFINITION

A **two-person constant-sum game** is a two-player game in which, for any choice of both player's strategies, the row player's reward and the column player's reward add up to a constant value c .

Of course, a two-person zero-sum game is just a two-person constant-sum game with $c = 0$. A two-person constant-sum game maintains the feature that the row and column players are in total conflict, because a unit increase in the row player's reward will always result in a unit decrease in the column player's reward. In general, the optimal strategies and value for a two-person constant-sum game may be found by the same methods used to find the optimal strategies and value for a two-person zero-sum game.

EXAMPLE 1

During the 8 to 9 P.M. time slot, two networks are vying for an audience of 100 million viewers. The networks must simultaneously announce the type of show they will air in that time slot. The possible choices for each network and the number of network 1 viewers (in millions) for each choice are shown in Table 5. For example, if both networks choose a western, the matrix indicates that 35 million people will watch network 1 and $100 - 35 = 65$ million people will watch network 2. Thus, we have a two-person constant-sum game with $c = 100$ (million). Does this game have a saddle point? What is the value of the game to network 1?

Solution

Looking at the row minima, we find that by choosing a soap opera, network 1 can be sure of at least $\max(15, 45, 14) = 45$ million viewers. Looking at the column maxima, we find that by choosing a western, network 2 can hold network 1 to at most $\min(45, 58, 70) = 45$ million viewers. Since

$$\max(\text{row minimum}) = \min(\text{column maximum}) = 45$$

we find that Equation (1) is satisfied. Thus, network 1's choosing a soap opera and network 2's choosing a western yield a saddle point; neither side will do better if it unilaterally changes strategy (check this). Thus, the value of the game to network 1 is 45 million viewers, and the value of the game to network 2 is $100 - 45 = 55$ million viewers. The

TABLE 5
A Constant-Sum Game

	Network 2			Row Minimum
	Western	Soap Opera	Comedy	
Western	35	15	60	15
Soap Opera	45	58	50	45
Comedy	38	14	70	14
Column Maximum	45	58	70	

optimal strategy for network 1 is to choose a soap opera, and the optimal strategy for network 2 is to choose a western.

Problems

Group A

- 1 Find the value and optimal strategy for the game in Table 6.

TABLE 6

2	2
1	3

- 2 Find the value and the optimal strategies for the two-person zero-sum game in Table 7.

TABLE 7

4	5	5	8
6	7	6	9
5	7	5	4
6	6	5	5

Group B

- 3 Mad Max wants to travel from New York to Dallas by the shortest possible route. He may travel over the routes shown in Table 8. Unfortunately, the Wicked Witch can block one road leading out of Atlanta and one road leading out of

Nashville. Mad Max will not know which roads have been blocked until he arrives at Atlanta or Nashville. Should Mad Max start toward Atlanta or Nashville? Which routes should the Wicked Witch block?

Group C

- 4 Explain why the reward for a saddle point must be the smallest number in its row and the largest number in its column. Suppose a reward is the smallest in its row and the largest in its column. Must that reward yield a saddle point? (*Hint:* Think about the idea of weak duality discussed in Chapter 6.)

TABLE 8

Route	Length of Route (miles)
New York-Atlanta	800
New York-Nashville	900
Nashville-St. Louis	400
Nashville-New Orleans	200
Atlanta-St. Louis	300
Atlanta-New Orleans	600
St. Louis-Dallas	500
New Orleans-Dallas	300

15.2 Two-Person Zero-Sum Games: Randomized Strategies, Domination, and Graphical Solution

In the previous section, we found that not all two-person zero-sum games have saddle points. We now discuss how to find the value and optimal strategies for a two-person zero-sum game that does not have a saddle point. We begin with the simple game of Odds and Evens.

EXAMPLE 2 Odds and Evens Two players (called Odd and Even) simultaneously choose the number of fingers (1 or 2) to put out. If the sum of the fingers put out by both players is odd, Odd wins \$1 from Even. If the sum of the fingers is even, Even wins \$1 from Odd. We consider the row player to be Odd and the column player to be Even. Determine whether this game has a saddle point.

Solution This is a zero-sum game, with the reward matrix shown in Table 9. Since $\max(\text{row minimum}) = -1$ and $\min(\text{column maximum}) = +1$, Equation (1) is not satisfied, and this game has no saddle point. All we know is that Odd can be sure of a reward of at least -1 , and Even can hold Odd to a reward of at most $+1$. Thus, it is unclear how to determine the value of the game and the optimal strategies. Observe that for any choice of strategies by both players, there is a player who can benefit by unilaterally changing his or her strategy. For example, if both players put out one finger, then Odd could have increased her reward from -1 to $+1$ by putting out two fingers. Thus, no choice of strategies by the player is stable. We now determine optimal strategies and the value for this game.

Randomized or Mixed Strategies

To progress further with the analysis of Example 2 (and other games without saddle points), we must expand the set of allowable strategies for each player to include **randomized strategies**. Until now, we have assumed that each time a player plays a game, the player will choose the same strategy. Why not allow each player to select a probability of playing each strategy? For Example 2, we might define

- x_1 = probability that Odd puts out one finger
- x_2 = probability that Odd puts out two fingers
- y_1 = probability that Even puts out one finger
- y_2 = probability that Even puts out two fingers

If $x_1 \geq 0$, $x_2 \geq 0$, and $x_1 + x_2 = 1$, (x_1, x_2) is a randomized, or mixed, strategy for Odd. For example, the mixed strategy $(\frac{1}{2}, \frac{1}{2})$ could be realized by Odd if she tossed a coin before each play of the game and put out one finger for heads and two fingers for tails. Similarly, if $y_1 \geq 0$, $y_2 \geq 0$, and $y_1 + y_2 = 1$, (y_1, y_2) is a mixed strategy for Even.

Any mixed strategy (x_1, x_2, \dots, x_m) for the row player is a **pure strategy** if any of the x_i equals 1. Similarly, any mixed strategy (y_1, y_2, \dots, y_n) for the column player is a pure strategy if any of the y_i equals 1. A pure strategy is a special case of a mixed strategy in which a player always chooses the same action. Recall from Section 15.1 that the game in

TABLE 9
Reward Matrix for
Odds and Evens

Row Player (Odd)	Column Player (Even)		Row Minimum
	1 Finger	2 Fingers	
1 Finger	-1	+1	-1
2 Fingers	+1	-1	-1
Column Maximum	+1	+1	

Table 10 had a value of 5 (corresponding to a saddle point), so the row player's optimal strategy could be represented as the pure strategy $(0, 0, 1)$, and the column player's optimal strategy could be represented as the pure strategy $(0, 1, 0)$.

We continue to assume that both players will play two-person zero-sum games in accordance with the basic assumption of Section 15.1. In the context of randomized strategies, the assumption (from the standpoint of Odd) may be stated as follows: Odd should choose x_1 and x_2 to maximize her expected reward under the assumption that Even knows the value of x_1 and x_2 .

It is important to realize that even though we assume that Even knows the values of x_1 and x_2 , on a particular play of the game, he is not assumed to know Odd's actual strategy choice until the instant the game is played.

Graphical Solution of Odds and Evens

Finding Odd's Optimal Strategy

With this version of the basic assumption, we can determine the optimal strategy for Odd. Since $x_1 + x_2 = 1$, we know that $x_2 = 1 - x_1$. Thus, any mixed strategy may be written as $(x_1, 1 - x_1)$, and it suffices to determine the value of x_1 . Suppose Odd chooses a particular mixed strategy $(x_1, 1 - x_1)$. What is Odd's expected reward against each of Even's strategies? If Even puts out one finger, Odd will receive a reward of -1 with probability x_1 and a reward of $+1$ with probability $x_2 = 1 - x_1$. Thus, if Even puts out one finger and Odd chooses the mixed strategy $(x_1, 1 - x_1)$, Odd's expected reward is

$$(-1)x_1 + (+1)(1 - x_1) = 1 - 2x_1$$

As a function of x_1 , this expected reward is drawn as line segment AC in Figure 1. Similarly, if Even puts out two fingers and Odd chooses the mixed strategy $(x_1, 1 - x_1)$, Odd's expected reward is

$$(+1)(x_1) + (-1)(1 - x_1) = 2x_1 - 1$$

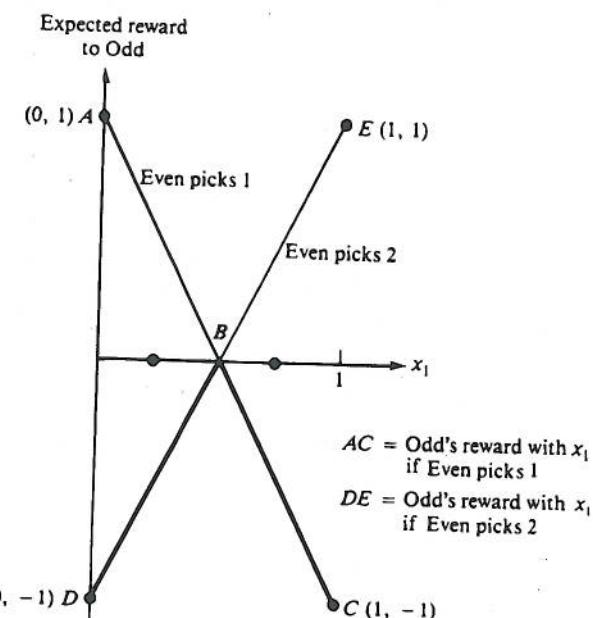
which is line segment DE in Figure 1.

Suppose Odd chooses the mixed strategy $(x_1, 1 - x_1)$. Since Even is assumed to know the value of x_1 , for any value of x_1 Even will choose the strategy (putting out one or two fingers) that yields a smaller expected reward for Odd. From Figure 1, we see that as a function of x_1 , Odd's expected reward will be given by the y coordinate in DBC . Since Odd wants to maximize her expected reward, she should choose the value of x_1 corresponding to point B . Point B occurs where the line segments AC and DE intersect, or where $1 - 2x_1 = 2x_1 - 1$. Solving this equation, we obtain $x_1 = \frac{1}{2}$. Thus, Odd should choose the mixed strategy $(\frac{1}{2}, \frac{1}{2})$. The reader should verify that against each of Even's strategies, $(\frac{1}{2}, \frac{1}{2})$ yields an expected reward of zero. Thus, zero is a **floor** on Odd's expected reward, because by choosing the mixed strategy $(\frac{1}{2}, \frac{1}{2})$, Odd can be sure that (for any choice of Even's strategy) her expected reward will always be at least zero.

TABLE 10

4	4	10
2	3	1
6	5	7

FIGURE 1
Choosing Odd's Strategy



Finding Even's Optimal Strategy

We now consider how Even should choose a mixed strategy (y_1, y_2) . Again, since $y_2 = 1 - y_1$, we may ask how Even should choose a mixed strategy $(y_1, 1 - y_1)$. The basic assumption implies that Even should choose y_1 to minimize his expected losses (or equivalently, minimize Odd's expected reward) under the assumption that Odd knows the value of y_1 . Suppose Even chooses the mixed strategy $(y_1, 1 - y_1)$. What will Odd do? If Odd puts out one finger, her expected reward is

$$(-1)y_1 + (+1)(1 - y_1) = 1 - 2y_1$$

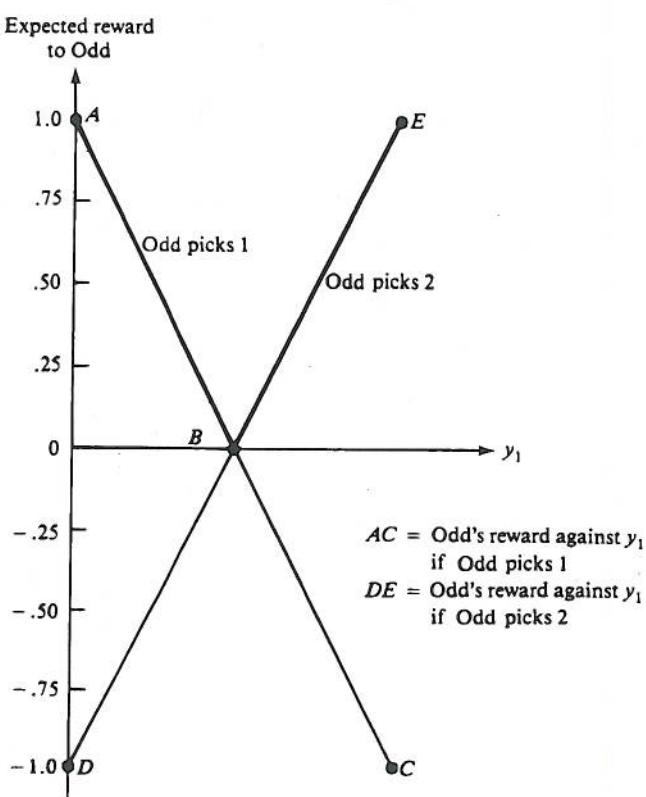
which is line segment AC in Figure 2. If Odd puts out two fingers, her expected reward is

$$(+1)(y_1) + (-1)(1 - y_1) = 2y_1 - 1$$

which is line segment DE in Figure 2. Since Odd is assumed to know the value of y_1 , she will put out the number of fingers corresponding to $\max(1 - 2y_1, 2y_1 - 1)$. Thus, for a given value of y_1 , Odd's expected reward (and Even's expected loss) will be given by the y -coordinate on the piecewise linear curve ABE .

Now Even chooses the mixed strategy $(y_1, 1 - y_1)$ that will make Odd's expected reward as small as possible. Thus, Even should choose the value of y_1 corresponding to the lowest point on ABE (point B). Point B is where the line segments AC and DE intersect, or where $1 - 2y_1 = 2y_1 - 1$, or $y_1 = \frac{1}{2}$. The basic assumption implies that Even should choose the mixed strategy $(\frac{1}{2}, \frac{1}{2})$. For this mixed strategy, Even's expected loss (and Odd's expected reward) is zero. We say that zero is a ceiling on Even's expected loss (or Odd's expected reward), because by choosing the mixed strategy $(\frac{1}{2}, \frac{1}{2})$, Even can ensure that his expected loss (for any choice of strategies by Odd) will not exceed zero.

FIGURE 2
Choosing Even's Strategy



More on the Idea of Value and Optimal Strategies

For the game of Odds and Evens, the row player's *floor* and the column player's *ceiling* are equal. This is not a coincidence. When each player is allowed to choose mixed strategies, the row player's floor will always equal the column player's ceiling. In Section 15.3, we use the Dual Theorem of Chapter 6 to prove this interesting result. We call the common value of the floor and ceiling the **value** of the game to the row player. Any mixed strategy for the row player that guarantees that the row player gets an expected reward at least equal to the value of the game is an **optimal strategy** for the row player. Similarly, any mixed strategy for the column player that guarantees that the column player's expected loss is no more than the value of the game is an optimal strategy for the column player. Thus, for Example 2, we have shown that the value of the game is zero, the row player's optimal strategy is $(\frac{1}{2}, \frac{1}{2})$, and the column player's optimal strategy is $(\frac{1}{2}, \frac{1}{2})$.

Example 2 illustrates that by allowing mixed strategies, we have enabled each player to find an optimal strategy in that *if the row player departs from her optimal strategy, the column player may have a strategy that reduces the row player's expected reward below the value of the game, and if the column player departs from his optimal strategy, the row player may have a strategy that increases her expected reward above the value of the game.* Table 11 illustrates this idea for the game of Odds and Evens.

TABLE 11
How to Make a Nonoptimal Strategy Pay the Price

Odd's Mixed Strategy	Even Can Choose	Odd's Expected Reward (Even's expected losses)
$x_1 < \frac{1}{2}$	2 fingers	< 0 (on BD in Figure 1)
$x_1 > \frac{1}{2}$	1 finger	< 0 (on BC in Figure 1)
Even's Mixed Strategy	Odd Can Choose	Odd's Expected Reward (Even's expected losses)
$y_1 < \frac{1}{2}$	1 finger	> 0 (on AB in Figure 2)
$y_1 > \frac{1}{2}$	2 fingers	> 0 (on BE in Figure 2)

For example, suppose that Odd chooses a nonoptimal mixed strategy with $x_1 < \frac{1}{2}$. Then by choosing two fingers, Even ensures that Odd's expected reward can be read from BD in Figure 1. This means that if Odd chooses a mixed strategy having $x_1 < \frac{1}{2}$, her expected reward can be negative (less than the value of the game).

To close this section, we find the value and optimal strategies for a more complicated game.

EXAMPLE 3

A fair coin is tossed, and the result is shown to player 1. Player 1 must then decide whether to pass or bet. If player 1 passes, he must pay player 2 \$1. If player 1 bets, player 2 (who does not know the result of the coin toss) may either fold or call the bet. If player 2 folds, she pays player 1 \$1. If player 2 calls and the coin comes up heads, she pays player 1 \$2; if player 2 calls and the coin comes up tails, player 1 must pay her \$2. Formulate this as a two-person zero-sum game. Then graphically determine the value of the game and each player's optimal strategy.

Solution

Player 1's strategies may be represented as follows: **PP**, pass on heads and pass on tails; **PB**, pass on heads and bet on tails; **BP**, bet on heads and pass on tails; and **BB**, bet on heads and bet on tails. Player 2 simply has the two strategies call and fold. For each choice of strategies, player 1's expected reward is as shown in Table 12.

To illustrate these computations, suppose player 1 chooses **BP** and player 2 calls. Then with probability $\frac{1}{2}$, heads is tossed. Then player 1 bets, is called, and wins \$2 from player 2. With probability $\frac{1}{2}$, tails is tossed. In this case, player 1 passes and pays player 2 \$1. Thus, if player 1 chooses **BP** and player 2 calls, player 1's expected reward is $(\frac{1}{2})(2) + (\frac{1}{2})(-1) = \0.50 . For each line in Table 12, the first term in the expectation corresponds to heads being tossed, and the second term corresponds to tails being tossed.

Example 3 may be described as the two-person zero-sum game represented by the reward matrix in Table 13. Since $\max(\text{row minimum}) = 0 < \min(\text{column maximum}) = \frac{1}{2}$, this game does not have a saddle point. Observe that player 1 would be unwise ever to choose the strategy **PP**, because (for each of player 2's strategies) player 1 could do better than **PP** by choosing either **BP** or **BB**. In general, a strategy i for a given player is dominated by a strategy i' if, for each of the other player's possible strategies, the given player does at least as well with strategy i' as he or she does with strategy i , and if for at least one of the other player's strategies, strategy i' is superior to strategy i . (This definition of a dominated strategy is similar to the definition of a dominated action given in Section 13.1.) A player may eliminate all dominated strategies from consideration. We have just shown that for player 1, **BP** or **BB** dominates **PP**. Similarly, the reader should be able to show

T A B L E 12
Computation of Reward
Matrix for Example 3

Player 1's Expected Reward		
PP vs. call	$(\frac{1}{2})(-1) + (\frac{1}{2})(-1) = -\1	
PP vs. fold	$(\frac{1}{2})(-1) + (\frac{1}{2})(-1) = -\1	
PB vs. call	$(\frac{1}{2})(-1) + (\frac{1}{2})(-2) = -\1.50	
PB vs. fold	$(\frac{1}{2})(-1) + (\frac{1}{2})(1) = \0	
BP vs. call	$(\frac{1}{2})(2) + (\frac{1}{2})(-1) = \0.50	
BP vs. fold	$(\frac{1}{2})(1) + (\frac{1}{2})(-1) = \0	
BB vs. call	$(\frac{1}{2})(2) + (\frac{1}{2})(-2) = \0	
BB vs. fold	$(\frac{1}{2})(1) + (\frac{1}{2})(1) = \1	

T A B L E 13
Reward Matrix for Example 3

Player 1	Player 2	Row Minimum
	Call Fold	
PP	-1 -1	-1
PB	$-\frac{3}{2}$ 0	$-\frac{3}{2}$
BP	$\frac{1}{2}$ 0	0
BB	0 1	0
Column Maximum	$\frac{1}{2}$ 1	

that player 1's PB strategy is dominated by BP or BB. After eliminating the dominated strategies PP and PB, we are left with the game matrix shown in Table 14.

As with Odds and Evens, this game has no saddle point, and we proceed with a graphical solution. Let

$$x_1 = \text{probability that player 1 chooses BP}$$

$$x_2 = 1 - x_1 = \text{probability that player 1 chooses BB}$$

$$y_1 = \text{probability that player 2 chooses call}$$

$$y_2 = 1 - y_1 = \text{probability that player 2 chooses fold}$$

To determine the optimal strategy for player 1, observe that for any value of x_1 , his expected reward against calling is

$$(\frac{1}{2})(x_1) + 0(1 - x_1) = \frac{x_1}{2}$$

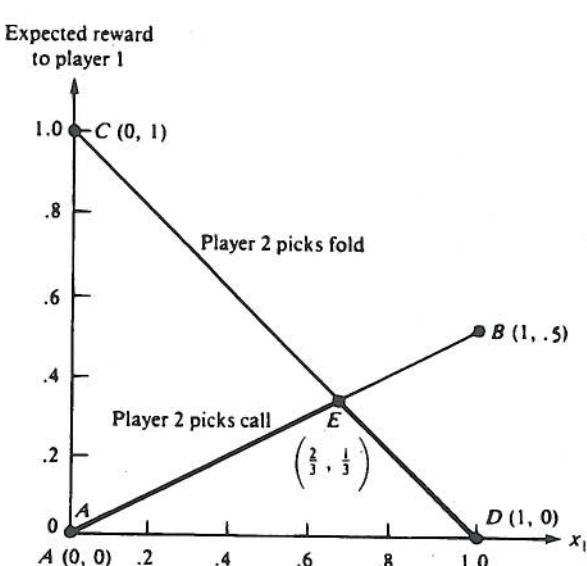
which is line segment AB in Figure 3. Against folding, player 1's expected reward is

$$0(x_1) + 1(1 - x_1) = 1 - x_1$$

T A B L E 14
Reward Matrix for Example 3
after Dominated Strategies
Have Been Eliminated

Player 1	Player 2	Row Minimum
	Call Fold	
BP	$\frac{1}{2}$ 0	0
BB	0 1	0
Column Maximum	$\frac{1}{2}$	1

FIGURE 3
How Player 1 Chooses Optimal Strategy in Example 3



which is line segment CD in Figure 3. Since player 2 is assumed to know the value of x_1 , player 1's expected reward (as a function of x_1) is given by the piecewise linear curve AED in Figure 3. Thus, to maximize his expected reward, player 1 should choose the value of x_1 corresponding to point E , which solves $x_1/2 = 1 - x_1$, or $x_1 = \frac{2}{3}$. Then $x_2 = 1 - \frac{2}{3} = \frac{1}{3}$, and player 1's expected reward against either of player 2's strategies is $\frac{x_1}{2}$ (or $1 - x_1$) = $\frac{1}{3}$.

How should player 2 choose y_1 ? (Remember, $y_2 = 1 - y_1$.) For a given value of y_1 , suppose player 1 chooses **BP**. Then his expected reward is

$$\left(\frac{1}{2}\right)(y_1) + 0(1 - y_1) = \frac{y_1}{2}$$

which is line segment AB in Figure 4. For a given value of y_1 , suppose player 1 chooses **BB**. Then his expected reward is

$$0(y_1) + 1(1 - y_1) = 1 - y_1$$

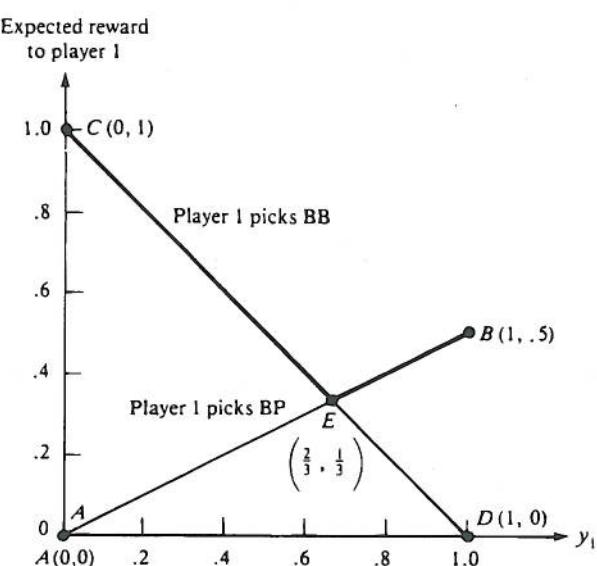
which is line segment CD in Figure 4. Thus, for a given value of y_1 , player 1 will choose a strategy that causes his expected reward to be given by the piecewise linear curve CED in Figure 4. Knowing this, player 2 should choose the value of y_1 corresponding to point E in Figure 4. The value of y_1 at point E is the solution to $\frac{y_1}{2} = 1 - y_1$, or $y_1 = \frac{2}{3}$ (and $y_2 = \frac{1}{3}$). The reader should check that no matter what player 1 does, player 2's mixed strategy $(\frac{2}{3}, \frac{1}{3})$ ensures that player 1 earns an expected reward of $\frac{1}{3}$.

In summary, the value of the game is $\frac{1}{3}$ to player 1; the optimal mixed strategy for player 1 is $(\frac{2}{3}, \frac{1}{3})$; and the optimal strategy for player 2 is also $(\frac{2}{3}, \frac{1}{3})$.

REMARKS

- 1 Observe that player 1 should bet $\frac{1}{3}$ of the time that he has a losing coin. Thus, our simple model indicates that player 1's optimal strategy includes bluffing.
- 2 In Problem 4 at the end of this section, it will be shown that if player 1 deviates from his optimal strategy, player 2 can hold him to an expected reward that is less than the value $(\frac{1}{3})$ of the game. Similarly, Problem 5 will show that if player 2 deviates from her optimal strategy, player 1 can earn an expected reward in excess of the value $(\frac{1}{3})$ of the game.

FIGURE 4
How Player 2 Chooses Optimal
Strategy in Example 3



3 Although we have only applied the graphical method to games in which each player (after dominated strategies have been eliminated) has only two strategies, the graphical approach can be used to solve two-person zero-sum games in which only one player has two strategies (games in which the reward matrix is $2 \times n$ or $m \times 2$). We choose, however, to solve all non- 2×2 two-person games by the linear programming method outlined in the next section.

Problems

Group A

1 Find the value and the optimal strategies for the two-person zero-sum game in Table 15.

2 Player 1 writes an integer between 1 and 20 on a slip of paper. Without showing this slip of paper to player 2, player 1 tells player 2 what he has written. Player 1 may lie or tell the truth. Player 2 must then guess whether or not player 1 has told the truth. If caught in a lie, player 1 must pay player 2 \$10; if falsely accused of lying, player 1 collects \$5 from player 2. If player 1 tells the truth and player 2 guesses that player 1 has told the truth, then player 1 must pay \$1 to player 2. If player 1 lies and player 2 does not guess that player 1 has lied, player 1 wins \$5 from player 2. Determine the value of this game and each player's optimal strategy.

TABLE 15

1	2	3
2	0	3

3 Find the value and optimal strategies for the two-person zero-sum game in Table 16.

TABLE 16

2	1	3
4	3	2

4 For Example 3, show that if player 1 deviates from his optimal strategy, then player 2 can ensure that player 1 earns an expected reward that is less than the value ($\frac{1}{3}$) of the game.

5 For Example 3, show that if player 2 deviates from her optimal strategy, then player 1 can ensure that he earns an expected reward that is more than the value ($\frac{1}{3}$) of the game.

6 Two competing firms must simultaneously determine how much of a product to produce. The total profit earned by the two firms is always \$1000. If firm 1's production level is low and firm 2's is also low, firm 1 earns a profit of \$500; if firm 1's level is low and 2's is high, firm 1's profit is \$400. If

firm 1's production level is high and so is firm 2's, then firm 1's profit is \$600, but if firm 1's level is high while firm 2's level is low, then firm 1's profit is only \$300. Find the value and optimal strategies for this constant-sum game.

- 7 Mo and Bo each have a quarter and a penny. Simultaneously they each display a coin. If the coins match Mo wins both coins; if they don't match Bo wins both coins. Determine optimal strategies for this game.

Group B

8 State University is about to play Ivy College for the state tennis championship. The State team has two players (A and B), and the Ivy team has three players (X, Y, and Z). The following facts are known about the players' relative abilities: X will always beat B; Y will always beat A; A will always beat Z. In any other match, each player has a $\frac{1}{2}$ chance of winning. Before State plays Ivy, the State coach must determine who will play first singles and who will play second singles. The Ivy coach (after choosing which two players will play singles) must also determine who will play first singles and second singles. Assume that each coach wants to maximize the expected number of singles matches won by the team. Use game theory to determine optimal strategies for each coach and the value of the game to each team.

- 9 Consider a two-person zero-sum game with the reward matrix in Table 17. Suppose this game does not have a saddle point. Show that the optimal strategy for the row player is to play the first row a fraction $(d - c)/(a + d - b - c)$ of the time and the optimal strategy for the column player is to play the first column a fraction $(d - b)/(a + d - b - c)$ of the time.

- 10 Consider the following simplified version of football. On each play the offense chooses to run or pass. At the same time, the defense chooses to play a run defense or pass defense. The number of yards gained on each play is

determined by the reward matrix in Table 18. The offense's goal is to maximize the average yards gained per play.

- a Use Problem 9 to show that the offense should run 10/17 of the time.

- b Suppose that the effectiveness of a pass against the run defense improves. Use the results of Problem 9 to show that the offense should pass less! Can you give an explanation for this strange phenomenon?

- 11 Use the idea of dominated strategies to determine optimal strategies for the reward matrix in Table 19.

TABLE 17

<i>a</i>	<i>b</i>
<i>c</i>	<i>d</i>

TABLE 18

Offense	Defense	
	Run	Pass
Run	1	8
Pass	10	0

TABLE 19

-5	-10	-1	-10	2	-1
-1	2	-10	7	-5	20
2	7	-5	-10	-10	7
7	20	-1	-1	-1	2
20	7	-10	7	-1	-10

15.3 Linear Programming and Zero-Sum Games

Linear programming can be used to find the value and optimal strategies (for the row and column players) for any two-person zero-sum game. To illustrate the main ideas, we consider the well-known game Stone, Paper, Scissors.

EXAMPLE 4 Stone, Paper, Scissors Each of two players simultaneously utters one of the three words *stone*, *paper*, or *scissors*. If both players utter the same word, the game is a draw. Otherwise, one player wins \$1 from the other player according to the following: Scissors defeats (cuts) paper, paper defeats (covers) stone, and stone defeats (breaks) scissors. Find the value and

optimal strategies for this two-person zero-sum game. The solution is given later in section.

The reward matrix is shown in Table 20. Observe that no strategies are dominated that the game does not have a saddle point. To determine optimal mixed strategies for row and the column player, define

- x_1 = probability that row player chooses stone
- x_2 = probability that row player chooses paper
- x_3 = probability that row player chooses scissors
- y_1 = probability that column player chooses stone
- y_2 = probability that column player chooses paper
- y_3 = probability that column player chooses scissors

The Row Player's LP

If the row player chooses the mixed strategy (x_1, x_2, x_3) , then her expected reward against each of the column player's strategies is as shown in Table 21. Suppose the row player chooses the mixed strategy (x_1, x_2, x_3) . By the basic assumption, the column player will choose a strategy that makes the row player's expected reward equal to $\min(x_2 - x_3, -x_1 + x_3, x_1 - x_2)$. Then the row player should choose (x_1, x_2, x_3) to make $\min(x_2 - x_3, -x_1 + x_3, x_1 - x_2)$ as large as possible. To obtain an LP formulation (called the row player's LP) that will yield the row player's optimal strategy, observe that for any values of x_1, x_2 , and x_3 , $\min(x_2 - x_3, -x_1 + x_3, x_1 - x_2)$ is just the largest number (call it v) that is simultaneous less than or equal to $x_2 - x_3, -x_1 + x_3$, and $x_1 - x_2$. After noting that x_1, x_2 , and x_3 must

TABLE 20
Reward Matrix for
Stone, Paper, Scissors

		Row Player			Column Player	Row Minimum
		Stone	Paper	Scissors		
Stone	0	-1	+1		-1	
	+1	0	-1		-1	
	-1	+1	0		-1	
Column Maximum		+1	+1	+1		

TABLE 21
Expected Reward to Row
Player in Stone, Paper, Scissors

Column Player Chooses	Row Player's Expected Reward If Row Player Chooses (x_1, x_2, x_3)
Stone	$x_2 - x_3$
Paper	$-x_1 + x_3$
Scissors	$x_1 - x_2$

ter in this

iated and
es for the

against
player
er will
 $-x_1 +$
 $-x_1 +$
r's LP)
and x_3 ,
eously
3 must

satisfy $x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$, and $x_1 + x_2 + x_3 = 1$, we see that the row player's optimal strategy can be found by solving the following LP:

$$\begin{aligned} \max z &= v \\ \text{s.t.} \quad v &\leq x_2 - x_3 && (\text{Stone constraint}) \\ v &\leq -x_1 + x_3 && (\text{Paper constraint}) \\ v &\leq x_1 - x_2 && (\text{Scissors constraint}) \\ x_1 + x_2 + x_3 &= 1 \\ x_1, x_2, x_3 &\geq 0; v \text{ urs} \end{aligned} \tag{2}$$

Note that there is a constraint in (2) for each of the column player's strategies. The value of v in the optimal solution to (2) is the row player's *floor*, because no matter what strategy (pure or mixed) is chosen by the column player, the row player is sure to receive an expected reward of at least v .

The Column Player's LP

How should the column player choose an optimal mixed strategy (y_1, y_2, y_3) ? Suppose the column player has chosen the mixed strategy (y_1, y_2, y_3) . For each of the row player's strategies, we may compute the row player's expected reward if the column player chooses (y_1, y_2, y_3) (see Table 22). Since the row player is assumed to know (y_1, y_2, y_3) , the row player will choose a strategy to ensure that she obtains an expected reward of $\max(-y_2 + y_3, y_1 - y_3, -y_1 + y_2)$. Thus, the column player should choose (y_1, y_2, y_3) to make $\max(-y_2 + y_3, y_1 - y_3, -y_1 + y_2)$ as small as possible. To obtain an LP formulation that will yield the column player's optimal strategies, observe that for any choice of (y_1, y_2, y_3) , $\max(-y_2 + y_3, y_1 - y_3, -y_1 + y_2)$ will equal the smallest number that is simultaneously greater than or equal to $-y_2 + y_3, y_1 - y_3$, and $-y_1 + y_2$ (call this number w). Also note that for (y_1, y_2, y_3) to be a mixed strategy, (y_1, y_2, y_3) must satisfy $y_1 + y_2 + y_3 = 1, y_1 \geq 0, y_2 \geq 0$, and $y_3 \geq 0$. Thus, the column player may find his optimal strategy by solving the following LP:

$$\begin{aligned} \min z &= w \\ \text{s.t.} \quad w &\geq -y_2 + y_3 && (\text{Stone constraint}) \\ w &\geq y_1 - y_3 && (\text{Paper constraint}) \\ w &\geq -y_1 + y_2 && (\text{Scissors constraint}) \\ y_1 + y_2 + y_3 &= 1 \\ y_1, y_2, y_3 &\geq 0; w \text{ urs} \end{aligned} \tag{3}$$

Observe that (3) contains a constraint corresponding to each of the row player's strategies. Also, the optimal objective function w for (3) is a *ceiling* on the column player's expected losses (or the row player's expected reward), because by choosing a mixed strategy

TABLE 22
Expected Reward to Row
Player in Stone, Paper, Scissors

Row Player Chooses	Row Player's Expected Reward If Column Player Chooses (y_1, y_2, y_3)
Stone	$-y_2 + y_3$
Paper	$y_1 - y_3$
Scissors	$-y_1 + y_2$

(y_1, y_2, y_3) that solves (3), the column player can ensure that his expected losses will be (against any of the row player's strategies) at most w .

Relation Between the Row and the Column Player's LPs

It is easy to show that the column player's LP is the dual of the row player's LP. Begin by rewriting the row player's LP (2) as

$$\begin{aligned} \max z &= v \\ \text{s.t.} & -x_2 + x_3 + v \leq 0 \\ & x_1 - x_3 + v \leq 0 \\ & -x_1 + x_2 + v \leq 0 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \geq 0; v \text{ urs} \end{aligned} \tag{4}$$

Let the dual variables for the constraints in (4) be y_1, y_2, y_3 , and w , respectively. We can now show that the dual of the row player's LP is the column player's LP. As in Section 6.5, we read the row player's LP across in Table 23 and find the dual of the row player's LP by reading down. Recall that the dual constraint corresponding to the variable v will be an equality constraint (because v is urs), and the dual variable w corresponding to the primal constraint $x_1 + x_2 + x_3 = 1$ will be urs (because $x_1 + x_2 + x_3 = 1$ is an equality constraint). Reading down in Table 23, we find the dual of the row player's LP (4) to be

$$\begin{aligned} \min z &= w \\ \text{s.t.} & y_2 - y_3 + w \geq 0 \\ & -y_1 + y_3 + w \geq 0 \\ & y_1 - y_2 + w \geq 0 \\ & y_1 + y_2 + y_3 = 1 \\ & y_1, y_2, y_3 \geq 0; w \text{ urs} \end{aligned}$$

After transposing all terms involving y_1, y_2 , and y_3 in the first three constraints to the right-hand side, we see that the last LP is the same as the column player's LP (3). Thus, the dual of the row player's LP is the column player's LP. (Of course, the dual of the column player's LP would be the row player's LP.)

It is easy to show that both the row player's LP (2) and the column player's LP (3) have an optimal solution (that is, neither LP can be infeasible or unbounded). Then the Dual Theorem of Section 6.7 implies that v , the optimal objective function value for the row player's LP, and w , the optimal objective function value for the column player's LP, are equal. Thus, the row player's floor equals the column player's ceiling. This result is often known as the **Minimax Theorem**. We call the common value of v and w the **value**

TABLE 23
Dual or Row Player's LP

	Min			Max	
y_1 ($y_1 \geq 0$)	x_1	x_2	x_3	v	
	0	-1	1	1	≤ 0
y_2 ($y_2 \geq 0$)	1	0	-1	1	≤ 0
y_3 ($y_3 \geq 0$)	-1	1	0	1	≤ 0
w (urs)	1	1	1	0	$= 1$
	≥ 0	≥ 0	≥ 0	= 1	

of the game to the row player. As in Sections 15.1 and 15.2, the row player can (by playing an optimal strategy) guarantee that her expected reward will at least equal the value of the game. Similarly, the column player can (by playing an optimal strategy) guarantee that his expected losses will not exceed the value of the game. It can also be shown (see Problem 6 at the end of this section) that the optimal strategies obtained via linear programming represent a stable equilibrium, because neither player can improve his or her situation by a unilateral change in strategy.

For the Stone, Paper, Scissors game, the optimal solution to the row player's LP (2) is $w = 0, x_1 = \frac{1}{3}, x_2 = \frac{1}{3}, x_3 = \frac{1}{3}$, and the optimal solution to the column player's LP (3) is $v = 0, y_1 = \frac{1}{3}, y_2 = \frac{1}{3}, y_3 = \frac{1}{3}$. Note that the first solution is feasible in (2), and the second solution is feasible in (3). Since each solution yields an objective function value of zero, Lemma 2 in Chapter 6 shows that $x_1 = \frac{1}{3}, x_2 = \frac{1}{3}, x_3 = \frac{1}{3}$ is optimal for the row player's LP, and $y_1 = \frac{1}{3}, y_2 = \frac{1}{3}, y_3 = \frac{1}{3}$ is optimal for the column player's LP.

The complementary slackness theory of linear programming (discussed in Section 6.10) could have been used to find the optimal strategies and value for Stone, Paper, Scissors (as well as other games). Before showing how, we state the row and the column player's LPs for a general two-person zero-sum game.

Consider a two-person zero-sum game with the reward matrix shown in Table 24. The reasoning used to derive (2) and (3) yields the following LPs:

$$\begin{aligned}
 & \max z = v \\
 \text{s.t. } & v \leq a_{11}x_1 + a_{21}x_2 + \cdots + a_{m1}x_m && \text{(Column 1 constraint)} \\
 & v \leq a_{12}x_1 + a_{22}x_2 + \cdots + a_{m2}x_m && \text{(Column 2 constraint)} \\
 & \vdots && \\
 & v \leq a_{1n}x_1 + a_{2n}x_2 + \cdots + a_{mn}x_m && \text{(Column } n \text{ constraint)} \\
 & x_1 + x_2 + \cdots + x_m = 1 && \\
 & x_i \geq 0 \quad (i = 1, 2, \dots, m); v \text{ urs} && \\
 \text{Row Player's LP} & & & \\
 & \min z = w \\
 \text{s.t. } & w \geq a_{11}y_1 + a_{12}y_2 + \cdots + a_{1n}y_n && \text{(Row 1 constraint)} \\
 & w \geq a_{21}y_1 + a_{22}y_2 + \cdots + a_{2n}y_n && \text{(Row 2 constraint)} \\
 & \vdots && \\
 & w \geq a_{m1}y_1 + a_{m2}y_2 + \cdots + a_{mn}y_n && \text{(Row } m \text{ constraint)} \\
 & y_1 + y_2 + \cdots + y_n = 1 && \\
 & y_j \geq 0 \quad (j = 1, 2, \dots, n); w \text{ urs} && \\
 \text{Column Player's LP} & & & \\
 \end{aligned} \tag{5}$$

In (5), x_i = probability that the row player chooses row i , and in (6), y_j = probability that the column player chooses column j . The j th constraint ($j = 1, 2, \dots, n$) in the row player's LP implies that her expected reward against column j must at least equal v ; otherwise, the column player could hold the row player's expected reward below v by choosing column j . Similarly, the i th ($i = 1, 2, \dots, m$) constraint in the column player's LP implies that if the row player chooses row i , then the column player's expected losses cannot exceed w .

TABLE 24
A General Two-Person Zero-Sum Game

	Row Player				Column Player			
	Strategy 1	Strategy 2	...	Strategy n				
Strategy 1	a_{11}	a_{12}	...	a_{1n}				
Strategy 2	a_{21}	a_{22}	...	a_{2n}				
⋮	⋮	⋮		⋮				
Strategy m	a_{m1}	a_{m2}	...	a_{mn}				

if this were not the case, the row player could obtain an expected reward that exceeded w by choosing row i .

How to Solve the Row and the Column Players' LPs

It is easy to show (see Problem 9 at the end of this section) that if we add a constant c to each entry in a game's reward matrix, the optimal strategies for each player remain unchanged, but the optimal values of w and v (and thus the value of the game) are both increased by c . Let A be the original reward matrix. Suppose we add $c = |\text{most negative entry in reward matrix}|$ to each element of A . Call the new reward matrix A' . A' is a two-person constant-sum game. (Why?) Let \bar{v} and \bar{w} be the optimal objective function values for the row and the column players' LPs for A , and let \bar{v}' and \bar{w}' denote the same quantities for the game A' . Since A' will have no negative rewards, $\bar{v}' \geq 0$ and $\bar{w}' \geq 0$ must hold. Thus, when solving the row and the column players' LPs for A' , we may assume that $v' \geq 0$ and $w' \geq 0$ and ignore v urs and w urs. Then the optimal strategies for A' will be identical to the optimal strategies for A , and the value of $A' = (\text{value of } A) + c$, or value of $A = (\text{value of } A') - c$.

In solving small games by hand, it is often helpful to use the constraint $x_1 + x_2 + \dots + x_m = 1$ to eliminate one of the x_i 's from the row player's LP and to use the constraint $y_1 + y_2 + \dots + y_n = 1$ to eliminate one of the y_i 's from the column player's LP. Then (as illustrated by Examples 5 and 6, which follow), the complementary slackness results of Section 6.10 can often be used to solve the row and the column player's LPs simultaneously.

Solution

Example 4—Stone, Paper, Scissors The most negative element in the Stone, Paper, Scissors reward matrix is -1 . Therefore, we add $|-1| = 1$ to each element of the reward matrix. This yields the constant-sum game shown in Table 25. The row player's LP is as follows:

$$\begin{aligned} & \max v' \\ \text{s.t.} \quad & v' \leq x_1 + 2x_2 \\ & v' \leq x_2 + 2x_3 \\ & v' \leq 2x_1 + x_3 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3, v' \geq 0 \end{aligned}$$

TABLE 25
Modified Reward Matrix
for Stone, Paper, Scissors

Row Player	Column Player		
	Stone	Paper	Scissors
Stone	1	0	2
Paper	2	1	0
Scissors	0	2	1

Substituting $x_3 = 1 - x_1 - x_2$ transforms the row player's LP into the following LP:

$$\begin{aligned} & \max v' \\ \text{s.t.} & \begin{aligned} & (a) v' - x_1 - 2x_2 \leq 0 && (y_1, \text{ or column 1, constraint}) \\ & (b) v' + 2x_1 + x_2 \leq 2 && (y_2, \text{ or column 2, constraint}) \\ & (c) v' - x_1 + x_2 \leq 1 && (y_3, \text{ or column 3, constraint}) \end{aligned} \\ & x_1, x_2, v' \geq 0 \end{aligned} \tag{7}$$

The column player's LP is as follows:

$$\begin{aligned} & \min w' \\ \text{s.t.} & \begin{aligned} & w' \geq y_1 + 2y_3 && (x_1, \text{ or row 1, constraint}) \\ & w' \geq 2y_1 + y_2 && (x_2, \text{ or row 2, constraint}) \\ & w' \geq 2y_2 + y_3 && (x_3, \text{ or row 3, constraint}) \\ & y_1 + y_2 + y_3 = 1 \\ & y_1, y_2, y_3, w' \geq 0 \end{aligned} \end{aligned}$$

Substituting $y_3 = 1 - y_1 - y_2$ transforms the column player's LP into the following LP:

$$\begin{aligned} & \min w' \\ \text{s.t.} & \begin{aligned} & (a) w' + y_1 + 2y_2 \geq 2 && (x_1, \text{ or row 1, constraint}) \\ & (b) w' - 2y_1 - y_2 \geq 0 && (x_2, \text{ or row 2, constraint}) \\ & (c) w' + y_1 - y_2 \geq 1 && (x_3, \text{ or row 3, constraint}) \end{aligned} \\ & y_1, y_2, w' \geq 0 \end{aligned} \tag{8}$$

Since Stone, Paper, Scissors appears to be a fair game, we might conjecture that $v = w = 0$. This would make $v' = w' = 0 + 1 = 1$. Let's try this and conjecture that constraints (7a) and (7b) are binding in the optimal solution to (7). If this is the case, solving (7a) and (7b) simultaneously (with $v' = 1$) yields $x_1 = x_2 = \frac{1}{3}$. Since $x_1 = \frac{1}{3}$, $x_2 = \frac{1}{3}$, $w' = 1$ satisfies (7c) with equality, we have obtained a feasible solution to the row player's LP. Suppose this solution is optimal for the row player's LP. Then by complementary slackness (see Section 6.10), $x_1 > 0$ and $x_2 > 0$ would imply that the first two dual constraints in (8) must be binding in the optimal solution to (8). Solving (8a) and (8b) simultaneously (using $w' = 1$) yields $y_1 = y_2 = \frac{1}{3}$, $w' = 1$. This solution is dual feasible. Thus, we have found a primal feasible and a dual feasible solution, both of which have the same objective function value and are optimal. Thus:

- 1 The value of Stone, Paper, Scissors is $v' - 1 = 0$.
- 2 The optimal strategy for the row player is $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.
- 3 The optimal strategy for the column player is $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

REMARKS

Suppose we had not been able to conjecture that $v' = w' = 1$. Then the row player's LP (7) would have had three unknowns (x_1 , x_2 , and v'), and we might have hoped that the optimal solution to (7) occurred where all three constraints (7a)–(7c) were binding. Solving (7a)–(7c) simultaneously yields $v' = 1$, $x_1 = x_2 = \frac{1}{3}$, $x_3 = 1 - \frac{2}{3} = \frac{1}{3}$. If this is the optimal solution to the row player's LP, then complementary slackness implies that constraints (8a)–(8c) must all be binding. Simultaneously solving (8a)–(8c) yields $w' = 1$, $y_1 = y_2 = \frac{1}{3}$, $y_3 = 1 - \frac{2}{3} = \frac{1}{3}$. Again we have obtained a primal

feasible point and a dual feasible point having the same objective function value, and both solutions must be optimal.

EXAMPLE 5 Find the value and optimal strategies for the two-person zero-sum game in Table 26.

Solution The game has no saddle point and no dominated strategies, so we set up the row and the column players' LPs. Since all entries in the reward matrix are nonnegative, we are sure that the value of the game is nonnegative. The row and the column players' LPs for this game are as follows:

$$\begin{aligned} & \max v \\ \text{s.t. } & v \leq 30x_1 + 60x_2 \\ & v \leq 40x_1 + 10x_2 \\ & v \leq 36x_1 + 36x_2 \\ & x_1 + x_2 = 1 \\ & x_1, x_2, v \geq 0 \end{aligned} \tag{9}$$

Substituting $x_2 = 1 - x_1$ into the row player's LP yields

$$\begin{aligned} & \max v \\ \text{s.t. } & (a) v + 30x_1 \leq 60 \quad (y_1, \text{ or column 1, constraint}) \\ & (b) v - 30x_1 \leq 10 \quad (y_2, \text{ or column 2, constraint}) \\ & (c) v \leq 36 \quad (y_3, \text{ or column 3, constraint}) \\ & x_1, v \geq 0 \end{aligned} \tag{9'}$$

Similarly, we find

$$\begin{aligned} & \min w \\ \text{s.t. } & (a) w \geq 30y_1 + 40y_2 + 36y_3 \\ & (b) w \geq 60y_1 + 10y_2 + 36y_3 \\ & y_1 + y_2 + y_3 = 1 \\ & y_1, y_2, y_3, w \geq 0 \end{aligned} \tag{10}$$

and substituting $y_3 = 1 - y_1 - y_2$ into the column player's LP yields

$$\begin{aligned} & \min w \\ \text{s.t. } & (a) w + 6y_1 - 4y_2 \geq 36 \quad (x_1, \text{ or row 1, constraint}) \\ & (b) w - 24y_1 + 26y_2 \geq 36 \quad (x_2, \text{ or row 2, constraint}) \\ & y_1, y_2, w \geq 0 \end{aligned} \tag{10'}$$

TABLE 26
Reward Matrix for Example 5

Row Player	Column Player			Row Minimum
	30	40	36	
	60	10	36	10
Column Maximum	60	40	36	

both solution.

ble 26.

row and the
we are sure
LPs for this

When using complementary slackness to solve an LP and its dual, it is usually easier to first examine the LP with the smaller number of variables. Thus, we first examine (9'). We assume that (9'a) and (9'b) are both binding in the optimal solution to the row player's LP. Then $v = 35$, $x_1 = \frac{5}{6}$, $x_2 = \frac{1}{6}$ would be the optimal solution to the row player's LP. This solution is feasible in the row player's LP and makes constraint (9'c) nonbinding. If this solution is optimal for the row player's LP, then complementary slackness implies that (10'a) and (10'b) must both be binding and $y_3 = 0$ must hold. This implies that $y_1 + y_2 = 1$, or $y_2 = 1 - y_1$. Trying $w = 35$ and substituting $y_2 = 1 - y_1$ in (10'a) and (10'b) yields $y_1 = y_2 = \frac{1}{2}$. Thus, we have found a feasible solution ($v = 35$, $x_1 = \frac{5}{6}$, $x_2 = \frac{1}{6}$) to the row player's LP and a feasible solution ($w = 35$, $y_1 = \frac{1}{2}$, $y_2 = \frac{1}{2}$, $y_3 = 0$) to the column player's LP, both of which have the same objective function value. We have therefore found the value of the game and the optimal strategy for each player.

In closing, we note that while the column player's third strategy is not dominated by column 1 or column 2, he should still never choose column 3. Why?

In the following two-person zero-sum game, the complementary slackness method does not yield optimal strategies.

E X A M P L E 6

Two players in the game of Two-Finger Morra simultaneously put out either one or two fingers. Each player must also announce the number of fingers that he believes his opponent has put out. If neither or both players correctly guess the number of fingers put out by the opponent, the game is a draw. Otherwise, the player who guesses correctly wins (from the other player) the sum (in dollars) of the fingers put out by the two players. If we let (i, j) represent the strategy of putting out i fingers and guessing the opponent has put out j fingers, the appropriate reward matrix is as shown in Table 27.

Solution

Again, this game has no saddle point and no dominated strategies. To ensure that the value of the game is nonnegative, we add 4 to each entry in the reward matrix. This yields the

T A B L E 27
Reward Matrix for
Two-Finger Morra

Row Player	Column Player				Row Minimum
	(1, 1)	(1, 2)	(2, 1)	(2, 2)	
(1, 1)	0	2	-3	0	-3
(1, 2)	-2	0	0	3	-2
(2, 1)	3	0	0	-4	-4
(2, 2)	0	-3	4	0	-3
Column Maximum		3	2	4	3

TABLE 28
Transformed Reward Matrix
for Two-Finger Morra

Row Player	Column Player			
	(1, 1)	(1, 2)	(2, 1)	(2, 2)
(1, 1)	4	6	1	4
(1, 2)	2	4	4	7
(2, 1)	7	4	4	0
(2, 2)	4	1	8	4

reward matrix in Table 28. For this game, the row player's and the column player's LPs are as follows (recall that the value for the original Two-Finger Morra game = $v' - 4$):

$$\max v'$$

Row Player's LP	s.t.	(a) $v' \leq 4x_1 + 2x_2 + 7x_3 + 4(1 - x_1 - x_2 - x_3)$	$(y_1 \text{ constraint})$
		(b) $v' \leq 6x_1 + 4x_2 + 4x_3 + (1 - x_1 - x_2 - x_3)$	$(y_2 \text{ constraint})$
		(c) $v' \leq x_1 + 4x_2 + 4x_3 + 8(1 - x_1 - x_2 - x_3)$	$(y_3 \text{ constraint})$
		(d) $v' \leq 4x_1 + 7x_2 + 4(1 - x_1 - x_2 - x_3)$	$(y_4 \text{ constraint})$
		$x_1, x_2, x_3, v' \geq 0$	

$$\max w'$$

Column Player's LP	s.t.	(a) $w' \geq 4y_1 + 6y_2 + y_3 + 4(1 - y_1 - y_2 - y_3)$	$(x_1 \text{ constraint})$
		(b) $w' \geq 2y_1 + 4y_2 + 4y_3 + 7(1 - y_1 - y_2 - y_3)$	$(x_2 \text{ constraint})$
		(c) $w' \geq 7y_1 + 4y_2 + 4y_3$	$(x_3 \text{ constraint})$
		(d) $w' \geq 4y_1 + y_2 + 8y_3 + 4(1 - y_1 - y_2 - y_3)$	$(x_4 \text{ constraint})$
		$y_1, y_2, y_3, w' \geq 0$	

An attempt to use complementary slackness to solve the row and the column players' LPs fails, because the optimal strategies for both players are degenerate. (Try complementary slackness and see what happens.) Using LINDO (or the simplex) to solve the LPs yields the following solutions: For the row player's problem, $v' = 4$, $x_1 = 0$, $x_2 = \frac{3}{5}$, $x_3 = \frac{2}{5}$, $x_4 = 0$ or $v' = 4$, $x_1 = 0$, $x_2 = \frac{4}{7}$, $x_3 = \frac{3}{7}$, $x_4 = 0$; for the column player's problem, $w' = 4$, $y_1 = 0$, $y_2 = \frac{3}{5}$, $y_3 = \frac{2}{5}$, $y_4 = 0$ or $w' = 4$, $y_1 = 0$, $y_2 = \frac{4}{7}$, $y_3 = \frac{3}{7}$, $y_4 = 0$. Since each player's LP has alternative optimal solutions, each player actually has an infinite number of optimal strategies. For example, for any c satisfying $0 \leq c \leq 1$, $x_1 = 0$, $x_2 = \frac{3c}{5} + \frac{4(1-c)}{7}$, $x_3 = \frac{2c}{5} + \frac{3(1-c)}{7}$, $x_4 = 0$ would be an optimal strategy for the row player. Of course, the value (to the row player) of the original Two-Finger Morra Game is $v' - 4 = 0$.

REMARKS

- Observe that both players have the same optimal strategies. (This is no accident; see Problem 5 at the end of this section.)
- Also note that if each player utilizes her optimal strategy, neither player will ever lose or win any money. This illustrates the fact that if both players follow the basic assumption of two-person zero-sum game theory, conservative play will generally result, because the assumption is analogous to the minimax decision criterion of Section 13.1.
- Finally, we see that if each player uses his or her optimal strategy, then a player never guesses the same number of fingers that he or she has actually put out. This fact is explained in Table 29.

Now suppose the row player chooses the optimal strategy $(0, \frac{3}{5}, \frac{2}{5}, 0)$. Then the column player only breaks even by playing $(1, 1)$ and loses an average of $\frac{1}{5}$ per play when he plays $(2, 2)$. Similarly, if the row player chooses the optimal strategy $(0, \frac{4}{7}, \frac{3}{7}, 0)$, the column player breaks even with $(2, 2)$.

TABLE 29
Expected Reward to Row Player

Row Plays Optimal Strategy	Expected Reward to Row	
	Column Plays (1, 1)	Column Plays (2, 2)
(0, $\frac{3}{5}$, $\frac{2}{5}$, 0)	$-2(\frac{3}{5}) + 3(\frac{2}{5}) = 0$	$3(\frac{3}{5}) - 4(\frac{2}{5}) = \frac{1}{5}$
(0, $\frac{4}{7}$, $\frac{3}{7}$, 0)	$-2(\frac{4}{7}) + 3(\frac{3}{7}) = \frac{1}{7}$	$3(\frac{4}{7}) - 4(\frac{3}{7}) = 0$

LPs are
):

and loses an average of $\frac{1}{7}$ per play when he plays (1, 1). Thus, putting out the same number of fingers as you guess cannot have a positive expected reward against the other player's optimal strategy.

The preceding discussion explains why the seemingly reasonable strategy $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ is not optimal for either player. For instance, if the column player chooses the strategy $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ and the row player plays the optimal strategy $(0, \frac{3}{5}, \frac{2}{5}, 0)$, the row player's expected reward may be computed as in Table 30. In this situation, the expected reward received by the row player is $-2(\frac{3}{20}) + 0(\frac{3}{20}) + 0(\frac{3}{20}) + 3(\frac{3}{20}) + 3(\frac{2}{20}) + 0(\frac{2}{20}) + 0(\frac{2}{20}) - 4(\frac{2}{20}) = \frac{1}{20}$. Another way to see this: Each time the column player chooses (1, 1), (1, 2), or (2, 1), the players break even, but on the plays for which the column player chooses (2, 2) the row player wins an average of $\frac{1}{5}$ unit. Thus, the row player's expected reward is $(\frac{1}{4})(\frac{1}{5}) = \frac{1}{20}$ unit.

4 In Odds and Evens and in Stone, Paper, Scissors, the optimal strategies may have been intuitively obvious, but the game of Two-Finger Morra shows that game theory can often yield subtle insights into how a two-person zero-sum game should be played.

TABLE 30
Expected Reward to Row Player If Column Player

Plays $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$

Row Chooses	Column Chooses	Reward to Row	Probability of Occurrence
(1, 2)	(1, 1)	-2	$(\frac{3}{5})(\frac{1}{4}) = \frac{3}{20}$
(1, 2)	(1, 2)	0	$(\frac{3}{5})(\frac{1}{4}) = \frac{3}{20}$
(1, 2)	(2, 1)	0	$(\frac{3}{5})(\frac{1}{4}) = \frac{3}{20}$
(1, 2)	(2, 2)	3	$(\frac{3}{5})(\frac{1}{4}) = \frac{3}{20}$
(2, 1)	(1, 1)	3	$(\frac{2}{5})(\frac{1}{4}) = \frac{2}{20}$
(2, 1)	(1, 2)	0	$(\frac{2}{5})(\frac{1}{4}) = \frac{2}{20}$
(2, 1)	(2, 1)	0	$(\frac{2}{5})(\frac{1}{4}) = \frac{2}{20}$
(2, 1)	(2, 2)	-4	$(\frac{2}{5})(\frac{1}{4}) = \frac{2}{20}$

Using LINDO or LINGO to Solve Two-Person Zero-Sum Games

To use LINDO to solve for the value and optimal strategies in a two-person zero-sum game simply type in either the row or column player's problem. If, for example you type in the row player's problem your optimal z -value is the value of the game; your optimal values of the decision variables are the row player's optimal strategies; and the absolute value of the Dual Prices are the column player's optimal strategies. By the way, since v is unrestricted in sign you should use the command FREE v after the END statement.

The following LINGO model (file Game.lng) can be used to solve for the value and optimal strategies for Two-Finger Morra (or any two-person zero-sum game).

Problem 5

use or win
wo-person
analogous

er guesses
ble 29.
mn player
. Similarly,
with (2, 2)

MODEL:

```

1]SETS:
2]ROWS/1..4/:X;
3]COLS/1..4/;
4]MATRIX(ROWS,COLS):REW;
5]ENDSETS
6)FOR(COLS(J):@SUM(ROWS(I):REW(I,J)*X(I))>V; );
7)@SUM(ROWS(I):X(I))=1;
8)MAX=V;
9)@FREE(V);
10)DATA:
11)REW=0,2,-3,0,
12)-2,0,0,3,
13)3,0,0,-4,
14)0,-3,4,0;
15)ENDDATA
16)END

```

In line 2 we define the rows of our reward matrix, associating row i with $X(i)$ = probability that the row player plays row i . In line 3 we define the columns of the reward matrix. In line 4 we create the reward matrix itself and define the reward $REW(i,j)$ to the row player when row i and column j are played. For each column j line 6 creates the constraint that $\sum_i REW(i,j)*X(i) \geq V$. In line 7 we ensure that the row player's probabilities sum to 1. Row 8 creates the objective function of $\max z = v$. Row 9 uses the @FREE statement to allow v to be negative. In rows 11 through 14 we input the reward matrix.

To use this model to solve for optimal strategies in any two-person zero-sum game, change the number of rows and columns and change the entries in the reward matrix. Remember that the Dual Prices yield the column player's optimal strategies.

Summary of How to Solve a Two-Person Zero-Sum Game

To close our discussion of two-person zero-sum games, we summarize a procedure that can be used to find the value and optimal strategies for any two-person zero-sum (or constant-sum) game.

Step 1 Check for a saddle point. If the game has no saddle point, go on to Step 2.

Step 2 Eliminate any of the row player's dominated strategies. Looking at the reduced matrix (dominated rows crossed out), eliminate any of the column player's dominated strategies. Now eliminate any of the row player's dominated strategies. Continue in this fashion until no more dominated strategies can be found. Now proceed to Step 3.

Step 3 If the game matrix is now 2×2 , solve the game graphically. Otherwise, solve the game by using the linear programming methods of this section.

Problems

Group A

- 1 A soldier can hide in one of five foxholes (1, 2, 3, 4, or 5) (see Figure 5). A gunner has a single shot and may fire at any of the four spots A, B, C, or D. A shot will kill a soldier if the soldier is in a foxhole adjacent to the spot where the shot was fired. For example, a shot fired at spot B will kill the soldier if he is in foxhole 2 or 3, while a shot fired at spot D will kill the soldier if he is in foxhole 4 or 5. Suppose the

gunner receives a reward of 1 if the soldier is killed and a reward of 0 if the soldier survives the shot.

FIGURE 5

- ① A ② B ③ C ④ D ⑤

- a Assuming this to be a zero-sum game, construct the reward matrix.
- b Find and eliminate all dominated strategies.
- c We are given that an optimal strategy for the soldier is to hide $\frac{1}{3}$ of the time in foxholes 1, 3, and 5. We are also told that for the gunner, an optimal strategy is to shoot $\frac{1}{3}$ of the time at A, $\frac{1}{3}$ of the time at D, and $\frac{1}{3}$ of the time at B or C. Determine the value of the game to the gunner.
- d Suppose the soldier chooses the following nonoptimal strategy: $\frac{1}{2}$ of the time, hide in foxhole 1; $\frac{1}{4}$ of the time, hide in foxhole 3; and $\frac{1}{4}$ of the time, hide in foxhole 5. Find a strategy for the gunner that ensures that his expected reward will exceed the value of the game.
- e Write down each player's LP and verify that the strategies given in part (c) are optimal strategies.
- 2 Find each player's optimal strategy and the value of the two-person zero-sum game in Table 31.

TABLE 31

4	5	1	4
2	1	6	3
1	0	0	2

- 3 Find each player's optimal strategy and the value of the two-person zero-sum game in Table 32.

TABLE 32

2	4	6
3	1	5

- 4 Two armies are advancing on two cities. The first army is commanded by General Custard and has four regiments; the second army is commanded by General Peabody and has three regiments. At each city, the army that sends more regiments to the city captures both the city and the opposing army's regiments. If both armies send the same number of regiments to a city, the battle at the city is a draw. Each army scores 1 point per city captured and 1 point per captured regiment. Assume that each army wants to maximize the difference between its reward and its opponent's reward. Formulate this situation as a two-person zero-sum game and solve for the value of the game and each player's optimal strategies.

Group B

- 5 A two-person zero-sum game with an $n \times n$ reward matrix A is a **symmetric** game if $A = -A^T$.
- a Explain why a game having $A = -A^T$ is called a symmetric game.

- b Show that a symmetric game must have a value of zero.
- c Show that if $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ is an optimal strategy for the row player, then $(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ is also an optimal strategy for the column player.

- d What examples discussed in this chapter are symmetric games? How could the results of this problem make it easier to solve for the value and optimal strategies of a symmetric game?

- 6 For a two-person zero-sum game with an $m \times n$ reward matrix, let $\bar{x} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_m)$ be a solution to the row player's LP and $\bar{y} = (\bar{y}_1, \bar{y}_2, \dots, \bar{y}_n)$ be a solution to the column player's LP. Show that if the row player departs from his optimal strategy, he cannot increase his expected reward against \bar{y} .

- 7 Interpret the complementary slackness conditions for the row and the column players' LPs.

- 8 Wivco has observed the daily production and the daily variable production costs of widgets at the New York City plant. The data in Table 33 have been collected. Wivco believes that daily production and daily variable production costs are related as follows: For some numbers a and b ,

$$\text{Daily production cost} = a + b(\text{daily production})$$

Wivco wants to find estimates of a and b (\hat{a} and \hat{b}) that minimize the maximum error (in absolute value) incurred in estimating daily production costs. For example, if Wivco chooses $\hat{a} = 3$ and $\hat{b} = 2$, the predicted daily costs are shown in Table 34. In this case, the maximum error would be \$3000. Formulate an LP that can be used to find the optimal estimates \hat{a} and \hat{b} .

TABLE 33

Day	Production	Variable Production Cost
1	4000	\$9,000
2	6000	\$12,000
3	7000	\$14,000
4	1000	\$5,000
5	3000	\$8,000

TABLE 34

Day	Predicted Cost	Absolute Error
1	\$11,000	\$2000
2	\$15,000	\$3000
3	\$17,000	\$3000
4	\$5,000	\$0
5	\$9,000	\$1000

- 9 Suppose we add a constant c to every element in a reward matrix A . Call the new game matrix A' . Show that A and

A' have the same optimal strategies and that value of $A' =$ (value of A) + c .

15.4 Two-Person Nonconstant-Sum Games

Most game-theoretic models of business situations are not constant-sum games, because it is unusual for business competitors to be in total conflict.

In this section, we briefly discuss the analysis of two-person nonconstant-sum games in which cooperation between the players is not allowed. We begin with a discussion of the famous Prisoner's Dilemma.

EXAMPLE 7

Prisoner's Dilemma Two prisoners who escaped and participated in a robbery have been recaptured and are awaiting trial for their new crime. Although they are both guilty, the Gotham City district attorney is not sure he has enough evidence to convict them. To entice them to testify against each other, the D.A. tells each prisoner the following: "If only one of you confesses and testifies against your partner, the person who confesses will go free while the person who does not confess will surely be convicted and given a 20-year jail sentence. If both of you confess, you will both be convicted and sent to prison for 5 years. Finally, if neither of you confesses, I can convict you both of a misdemeanor and you will each get 1 year in prison." What should each prisoner do?

Solution

If we assume that the prisoners cannot communicate with each other, the strategies and rewards for each are as shown in Table 35. The first number in each cell of this matrix is the reward (negative, since years in prison is undesirable) to prisoner 1, and the second matrix in each cell is the reward to prisoner 2. It is important to note that the sum of the rewards in each cell varies from a high of $-2(-1 - 1)$ to a low of $-20(-20 + 0)$. Thus, this is not a constant-sum two-player game.

TABLE 35
Reward Matrix for
Prisoner's Dilemma

		Prisoner 2	
		Confess	Don't Confess
Prisoner 1	Confess	(-5, -5)	(0, -20)
	Don't confess	(-20, 0)	(-1, -1)

Suppose each prisoner seeks to eliminate any dominated strategies from consideration. For each prisoner, the "confess" strategy dominates the "don't confess" strategy. If each prisoner follows his undominated ("confess") strategy, however, each prisoner will spend 5 years in jail. On the other hand, if each prisoner chooses the dominated "don't confess" strategy, each prisoner will spend only 1 year in prison. Thus, if each prisoner chooses his dominated strategy, both are better off than if each prisoner chooses his undominated strategy.

DEFINITION

As in a two-person zero-sum game, a choice of strategy by each player (prisoner) is an **equilibrium point** if neither player can benefit from a unilateral change in strategy.

Thus, $(-5, -5)$ is an equilibrium point, because if either prisoner changes his strategy, his reward decreases (from -5 to -20). Clearly, however, each prisoner is better off at the point $(-1, -1)$. To see that the outcome $(-1, -1)$ may not occur, observe that $(-1, -1)$ is not an equilibrium point, because if we are currently at the outcome $(-1, -1)$, either prisoner can increase his reward (from -1 to 0) by changing his strategy from "don't confess" to "confess" (that is, each prisoner can benefit from double-crossing his opponent). This illustrates an important aspect of the Prisoner's Dilemma type of game: If the players are cooperating (if each prisoner chooses "don't confess"), each player can gain by double-crossing his opponent (assuming his opponent's strategy remains unchanged). If both players double-cross each other, however, they will both be worse off than if they had both chosen their cooperative strategy. This anomaly cannot occur in a two-person constant-sum game. (Why not?)

More formally, a Prisoner's Dilemma game may be described as in Table 36, where

- NC = noncooperative action
- C = cooperative action
- P = punishment for not cooperating
- S = payoff to person who is double-crossed
- R = reward for cooperating if both players cooperate
- T = temptation for double-crossing opponent

In a Prisoner's Dilemma game, (P, P) is an equilibrium point. This requires $P > S$. For (R, R) not to be an equilibrium point requires $T > R$. (This gives each player a temptation to double-cross his opponent.) The game is reasonable only if $R > P$. Thus, for Table 36 to represent a Prisoner's Dilemma game, we require that $T > R > P > S$. The Prisoner's Dilemma game is of interest because it explains why two adversaries often fail to cooperate with each other. This is illustrated by Examples 8 and 9.

TABLE 36
A General Prisoner's
Dilemma Reward Matrix

		Player 1	Player 2
		NC	C
NC	NC	(P, P)	(T, S)
	C	(S, T)	(R, R)

EXAMPLE 8

Hot Dog King and Hot Dog Chef, competing restaurants, are attempting to determine their advertising budgets for next year. The two restaurants will have combined sales of \$240 million and can spend either \$6 million or \$10 million on advertising. If one restaurant spends more money than the other, the restaurant that spends more money will have sales of \$190 million. If both companies spend the same amount on advertising, they will have equal sales. Each dollar of sales yields 10¢ of profit. Suppose each restaurant is interested

in maximizing (contribution of sales to profit) — (advertising costs). Find an equilibrium point for this game.

Solution

The appropriate reward matrix is shown in Table 37. If we identify spending \$10 million on advertising as the noncooperative action and spending \$6 million as the cooperative action, then (2, 2) (corresponding to heavy advertising by both restaurants) is an equilibrium point. Although both restaurants are better off at (6, 6) than at (2, 2), (6, 6) is unstable because either restaurant may gain by changing its strategy. Thus, to protect its market share, each restaurant must spend heavily on advertising.

TABLE 37
Reward Matrix for Advertising Game

		Hot Dog King	Hot Dog Chef
		Spend \$10 Million	Spend \$6 Million
Spend \$10 million	Spend \$10 million	(2, 2)	(9, -1)
	Spend \$6 million	(-1, 9)	(6, 6)

EXAMPLE 9

The Vulcans and the Klingons are engaged in an arms race in which each nation is assumed to have two possible strategies: develop a new missile or maintain the status quo. The reward matrix is assumed to be as shown in Table 38. This reward matrix is based on the assumption that if only one nation develops a new missile, the nation with the new missile will conquer the other nation. In this case, the conquering nation earns a reward of 20 units and the conquered nation loses 100 units. It is also assumed that the cost of developing a new missile is 10 units. Identify an equilibrium point for this game.

TABLE 38
Reward Matrix for Arms Race Game

		Vulcans	Klingons
		Develop New Missile	Maintain Status Quo
Develop new missile	Develop new missile	(-10, -10)	(10, -100)
	Maintain status quo	(-100, 10)	(0, 0)

Solution

Identifying "develop" as the noncooperative action and "maintain" as the cooperative action, we see that (-10, -10) (both nations choosing their noncooperative action) is an equilibrium point. Although (0, 0) leaves both nations better off than (-10, -10), we see that in this situation, each nation can gain from a double-cross. Thus, (0, 0) is not stable. This example shows how maintaining the balance of power may lead to an arms race.

The following two-person nonconstant-sum game is not a Prisoner's Dilemma game.

EXAMPLE 10 Angry Max drives toward James Bound on a deserted road. Each person has two strategies: swerve or don't swerve. The reward matrix in Table 39 needs no explanation! Find the equilibrium point(s) for this game.

TABLE 39
Reward Matrix for Swerve Game

		Angry Max		James Bound	
		Swerve	Don't Swerve	Swerve	Don't Swerve
Swerve	(0, 0)	(-5, 5)	(5, -5)	(-100, -100)	
	(5, -5)	(-100, -100)			

Solution For both $(5, -5)$ and $(-5, 5)$, neither player can gain by a unilateral change in strategy. Thus, $(5, -5)$ and $(-5, 5)$ are both equilibrium points.

Like constant-sum games, a nonconstant-sum game may fail to have an equilibrium point in pure strategies. It can be shown that if mixed strategies are allowed, then in any two-person nonconstant-sum game, each player has an equilibrium strategy (in that if one player plays her equilibrium strategy, the other player cannot benefit by deviating from her equilibrium strategy) (see Owen (1982, p.127)). For example, consider the two-person nonconstant-sum game in Table 40. For this game, the reader should verify that there is no equilibrium in pure strategies and also that each player's choosing the mixed strategy $(\frac{1}{2}, \frac{1}{2})$ is an equilibrium, because neither player can benefit from a unilateral change in strategy (see Problem 4 at the end of this section). Owen (1982, Chapter 7) discusses two-person nonconstant-sum games in which the players are allowed to cooperate.

TABLE 40
A Game with No Equilibrium
in Pure Strategies

		Player 1		Player 2	
		Strategy 1	Strategy 2	Strategy 1	Strategy 2
Strategy 1	(2, -1)	(-2, 1)	(-2, 1)	(2, -1)	
	(-2, 1)	(2, -1)			

Problems

Group A

- 1 Find an equilibrium point (if one exists in pure strategies) for the two-person nonconstant-sum game in Table 41.

TABLE 41

(9, -1)	(-2, -3)
(8, 7)	(-9, 11)

- 2 Find an equilibrium point in pure strategies (if any exists) for the two-person nonconstant-sum game in Table 42.

- 3 The New York City Council is ready to vote on two bills that authorize the construction of new roads in Manhattan

TABLE 42

(9, 9)	(-10, 10)
(10, -10)	(-1, 1)