12-1999

Algorithms for solving linear integer progamming problems

Lisa Catherine Watkins

To the Graduate Council:

I am submitting herewith a thesis written by Lisa Catherine Watkins entitled "Algorithms for solving linear integer progamming problems." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Yueh-er Kuo, Major Professor

We have read this thesis and recommend its acceptance:

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Lisa Catherine Watkins entitled "Algorithms For Solving Linear Integer Programming Problems." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.

Yueh-er Kuo

Yueh-er Kuo, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor and
Dean of The Graduate School

ALGORITHMS FOR SOLVING LINEAR INTEGER PROGRAMMING PROBLEMS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Lisa Catherine Watkins

December 1999

DEDICATION

This thesis is dedicated to my Dad,

Mr. Frank Moore Watkins

(1927 - 1993)

ACKNOWLEDGMENTS

The author is grateful to Dr. Yueh-er Kuo for her help and advice in the preparation of this thesis. Appreciation is also extended to committee members Dr. William Wade and Dr. Charles Collins for the time in reading this thesis.

The author would also like to thank her family for their love and support.

ABSTRACT

This thesis discusses several methods used to solve linear integer programming problems. Chapters 1 – 3 give the necessary linear programming background. Chapter 4 introduces integer programming and describes, in general, the two classes of solution methods – enumeration and cutting planes. Chapter 5 details two specific cutting plane methods, one all-integer approach and one fractional approach. Chapter 6 describes the branch-and-bound method, one of the enumeration methods. In Chapter 7 the additive algorithm for 0-1 programming problems is discussed. Chapter 8 describes the branch-and-cut method, a combination of the cutting plane and branch-and-bound approaches. Chapter 9 presents the plant location problem as an example of the integer programming problem. Examples of each method are included in the thesis.

TABLE OF CONTENTS

SECTION I: THEORY

CHAPTER PAGE

1 LINEAR PROGRAMMING 1

1.1 Introduction 1

1.2 Standard Form of the LP Problem 4

1.3 Properties of a Solution to the LP Problem 6

2 THE SIMPLEX METHOD 10

2.1 Basic Method 10

2.2 Addition of Slack Variables 15

2.3 Artificial Basis Technique 17

3 DUAL SIMPLEX METHOD 23

3.1 Introduction 23

3.2 The Dual Simplex Method 25

4 INTEGER PROGRAMMING 31

4.1 Introduction 31

4.2 Summary of Solution Methods 32

SECTION II: METHODS

CHAPTER PAGE

5 CUTTING PLANES 38

5.1 Introduction 38

5.2 Gomory All-Integer Method 39

5.3 Gomory Mixed-Integer Method 50

6 BRANCH-AND-BOUND 61

6.1 Introduction 61

6.2 Method Using LP Relaxation 65

6.3 Branching 71

6.4 Node Selection 73

6.5 Preprocessing 75

7 ADDITIVE ALGORITHM 79

7.1 Introduction 79

7.2 Negative Coefficients in the Objective Function 84

7.3 Balas' Additive Algorithm 85

7.4 General Branching 95

8 BRANCH-AND-CUT 100

8.1 Introduction 100

CHAPTER PAGE

8.2 Branch-and-Cut Algorithm 102

8.3 Cut Management 109

SECTION III: APPLICATION

9 PLANT LOCATION PROBLEM 113

9.1 Fixed-Charge Problem 113

9.2 Plant Location Problem 116

BIBLIOGRAPHY 123

APPENDIX 127

VITA 141

LIST OF TABLEAUS

TABLEAU PAGE

2-1 Initial step of simplex method 10

2-2 Initial step of Example 2.1 14

2-3 Vector $\mathbf{P}_2$ replaces vector $\mathbf{P}_3$ 15

2-4 Initial tableau of Example 2.2 16

2-5 Vector $\mathbf{P}_3$ replaces vector $\mathbf{P}_6$ 16

2-6 Initial tableau for artificial basis technique 19

2-7 Initial step of Example 2.3 21

2-8 Vector $\mathbf{P}_3$ replaces vector $\mathbf{P}_7$ 21

3-1 Initial step of Example 3.1 29

3-2 Vector $\mathbf{P}_1$ replaces vector $\mathbf{P}_4$ 30

5-1 Initial tableau with constraint row and column added 41

5-2 Initial tableau for Example 5.1 46

5-3 Second step of Example 5.1 47

5-4 Simplified tableau form 48

5-5 Third step 48

5-6 Fourth step 49

5-7 Optimal solution 49

TABLEAU PAGE

5-8 Initial tableau for Example 5.2 59

5-9 Optimal LP solution 59

5-10 Constraint added to tableau 60

5-11 Optimal mixed-integer solution 60

8-1 Optimal LP solution for Example 8.1 104

8-2 Optimal LP solution for Node 1 104

8-3 Optimal integer solution for Node 1 105

8-4 Optimal LP solution for Node 2 105

8-5 LP solution after first cutting plane added 107

8-6 Optimal integer solution for Node 3 107

LIST OF FIGURES

FIGURE PAGE

4-1 Feasible region for Example 4.1 33

4-2 Graphical example of cutting plane method 36

4-3 Graphical example of branch-and-bound method 37

6-1 Tree of solutions for 0-1 programming problem 64

6-2 Tree of solutions for initial branch on x_1 70

6-3 Tree of solutions for initial branch on x_2 71

7-1 Network of solutions for a 4 variable 0-1 problem 87

7-2 Network of solutions in which $x_4 = 1$ are eliminated 88

7-3 Example of a binary search tree 92

7-4 Search tree for Example 7.1 94

7-5 Flowchart for additive algorithm with general branching 97

8-1 Branch-and-cut tree before cutting 108

8-2 Branch-and-cut tree after cutting 108

9-1 Shipping costs from city *i* to region *j* 118

9-2 Summary of demand and capacity 120

FIGURE PAGE

A-1 LINDO Model Window 129

A-2 LINDO Solver Status Window 130

A-3 LINDO Reports Window 132

A-4 Model window for plant location problem 134

A-5 Reports window for plant location problem 135

A-6 What's Best model 137

A-7 Solver dialog box 139

A-8 Solver Results dialog box 139

A-9 Answer Report spreadsheet 140

SECTION I: THEORY

CHAPTER 1

LINEAR PROGRAMMING

1.1 Introduction

Linear Programming (LP) should not be confused with computer programming. The term programming in LP is used in the general sense of devising a plan or strategy. LP is a mathematical technique and does not necessarily involve computers. In practice, however, problems are usually solved on a computer due to the number of calculations required to solve even small LP models. The mathematical technique for solving LP problems was developed by George Dantzig in 1947 to solve planning problems in the U.S. Air Force [5]. LP is now used in many fields, including economics, engineering, agriculture, business, transport, and manufacturing.

A linear programming problem is an optimization problem consisting of three parts:

1. A linear function (the objective function) of decision variables (say, x_1, x_2, ..., x_n) that is to be maximized or minimized. The

decision variables are activities for which the decision-maker wants to select the values. The coefficients of the decision variables in the objective function will be called the *cost coefficients* and will be denoted by c_j, where $j = 1, 2, ..., n$. Although, in general, an LP problem may have multiple objective functions, the problems we will work with have a single objective function. The objective may be to maximize profit, minimize costs, or minimize distance traveled, to name a few.

2. A set of restrictions or constraints (each of which must be a linear equality or linear inequality) that restrict the values that may be assumed by the decision variables. We will call the coefficients of these restrictions the *constraint coefficients* and will denote them by a_{ij}, where $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$. The range of constraints that can be represented in an LP model may include

   - limits on the availability of resources such as land, labor, or finance
   - technical constraints representing, for example, machinery work rates, and
   - logical constraints such as specifying that the amount of a product sold must not exceed the amount produced.

3. The *sign restrictions,* which specify for each decision variable x_j either (1) variable x_j must be nonnegative – $x_j \geq 0$; or (2) variable x_j may be positive, negative, or zero – x_j is unrestricted in sign.

Linearity in the objective function and in the constraints has two implications:

1. Proportionality Assumption of LP. The contribution to the objective function from each decision variable is proportional to the value of the decision variable. Similarly, the contribution of each variable to the left-hand side of each constraint is proportional to the value of the variable.
2. Additivity Assumption of LP. The contribution to the objective function and to the left-hand side of each constraint is independent of the values of the other decision variables. This implies that the value of the objective function the left-hand side of each constraint is the sum of the contributions from each decision variable.

A third assumption of LP problems is the Divisibility Assumption, which requires that each decision variable be allowed to assume fractional values. If some or all of the variables in an LP problem must be nonnegative integers, then the problem is an integer programming problem. These problems will be discussed in Chapter 4 of this section.

1.2 Standard Form of the LP Problem

Written mathematically, the standard form of the general LP problem is to find a vector $(x_1, x_2, \ldots, x_n)$ which minimizes the linear form (i.e., the objective function)

$$c_1x_1 + c_2x_2 + \cdots + c_jx_j + \cdots + c_nx_n \qquad (1.1)$$

subject to the linear constraints

$$\begin{array}{l} a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n = b_1 \\ \quad \vdots \qquad\quad \vdots \qquad\quad \vdots \quad\;\; \vdots \\ a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n = b_i \\ \quad \vdots \qquad\quad \vdots \qquad\quad \vdots \quad\;\; \vdots \\ a_{m1}x_1 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n = b_m \end{array} \qquad (1.2)$$

and

$$x_j \geq 0, \; j = 1, 2, \ldots, n \qquad (1.3)$$

where the a_{ij}, b_i, and c_j are constants and $m < n$. We will always assume that Equations (1.2) have been multiplied by −1 where necessary to make all $b_i \geq 0$. The general LP problem can be stated in many forms due to the variety of notation in use. A few of the more common forms that will be used in this thesis are:

a. Minimize

$$\sum_{j=1}^{n} c_jx_j$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, 2, \ldots, m$$

and

$$x_j \geq 0, \quad j = 1, 2, \ldots, n$$

b. Minimize

$$\mathbf{cX}$$

subject to

$$\mathbf{AX} = \mathbf{b}$$

and

$$\mathbf{X} \geq \mathbf{0}$$

where $\mathbf{c} = (c_1, \ldots, c_n)$ is a row vector, $\mathbf{X} = (x_1, \ldots, x_n)$ is a column vector, $\mathbf{A} = (a_{ij})$, $\mathbf{b} = (b_1, \ldots, b_m)$ is a column vector, and $\mathbf{0}$ is an n-dimensional column vector of zeroes.

c. Minimize

$$\mathbf{cX}$$

subject to

$$x_1\mathbf{P}_1 + x_2\mathbf{P}_2 + \cdots + x_n\mathbf{P}_n = \mathbf{P}_0$$

and

$$\mathbf{X} \geq \mathbf{0}$$

where $\mathbf{P}_j$ for $j = 1, 2, \ldots, n$ is the j^{th} column of the matrix $\mathbf{A}$ and $\mathbf{P}_0 = \mathbf{b}$.

1.3 Properties of a Solution to the LP Problem

In this section we will state a number of standard definitions and theorems relating to the solution to the general LP problem. Proofs of theorems will be omitted here but can be found in linear programming texts such as Dantzig [4] and Gass [6].

Definition 1.1. A *feasible solution* to the LP problem is a vector $\mathbf{X} = (x_1, x_2, ..., x_n)$ which satisfies the constraints (1.2) and the nonnegativity conditions (1.3).

Definition 1.2a. A *basic solution* to (1.2) is a solution obtained by setting *n-m* variables equal to zero and solving for the remaining *m* variables, provided that the determinant of the coefficients of these *m* variables is nonzero. The *m* variables are called *basic variables.*

Definition 1.2b. A *basic feasible solution* is a basic solution which also satisfies the nonnegativity conditions(1.3).

Definition 1.3. A *nondegenerate basic feasible solution* is a basic feasible solution with exactly *m* positive x_i, that is, all basic variables are positive.

Definition 1.4. A *minimum feasible solution* is a feasible solution which also minimizes(1.1).

Definition 1.5. An *optimal basic feasible solution* is a basic solution that satisfies (1.1), (1.2), and(1.3).

Unless otherwise stated, when we refer to a solution, we will mean any feasible solution.

Theorem 1.1. The set of all feasible solutions (also called the feasible region) to the LP problem is a convex set. We will denote this convex set by **C**.

By Theorem 1.1, if a problem has more than one solution, it has an infinite number of solutions. Our task is to determine the one solution which minimizes the corresponding objective function. The amount of work needed to find this solution is somewhat simplified by Theorem 1.2.

Theorem 1.2. The objective function (1.1) assumes its minimum at an extreme point of the convex set **C** generated by the set of feasible solutions to the LP problem. If it assumes its minimum at more than one extreme point, then it takes on the same value for every convex combination of those particular points.

Recall that a feasible solution is a vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ with all $x_i \geq 0$ such that

$$x_1\mathbf{P}_1 + x_2\mathbf{P}_2 + \cdots + x_n\mathbf{P}_n = \mathbf{P}_0$$

Assume we have found a set of k vectors that is linearly independent and that there exists a nonnegative combination of these vectors that is equal to $\mathbf{P}_0$. Let this set of vectors be $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$. Then we have Theorem 1.3.

Theorem 1.3. If a set of $k \leq m$ vectors $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$ can be found that is linearly independent and such that

$$x_1\mathbf{P}_1 + x_2\mathbf{P}_2 + \cdots + x_n\mathbf{P}_n = \mathbf{P}_0$$

and all $x_i \geq 0$, then the point $\mathbf{X} = (x_1, x_2, \ldots, x_k, 0, \ldots, 0)$ is an extreme point solution of the convex set **C** of feasible solutions. Here **X** is an n-dimensional vector whose last n-k elements are zero.

Theorem 1.4. If $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ is an extreme point of **C**, then the vectors associated with positive x_i form a linearly independent set. From this it follows that, at most, m of the x_i are positive.

Corollary 1.1. Associated with every extreme point of **C** is a set of m linearly independent vectors from the given set $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n$.

Theorem 1.5. $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ is an extreme point of **C** if and only if the positive x_j are coefficients of the linearly independent vectors $\mathbf{P}_j$ in

$$\sum_{j=1}^{n} x_j \mathbf{P}_j = \mathbf{P}_0$$

As a result of the theorems in this section we have:

1. There is an extreme point of **C** at which the objective function takes on its minimum.
2. Every basic feasible solution corresponds to an extreme point of **C**.
3. Every extreme point of **C** has m linearly independent vectors of the given set of n associated with it.

From the above we can conclude that we need only investigate extreme point solutions and hence only those feasible solutions generated by m linearly independent vectors. Since there are at most

$\binom{n}{m}$ sets of m linearly independent vectors from the given set of n, the value $\binom{n}{m}$ is an upper bound to the number of possible solutions to the problem. For large n and m it would be impossible to evaluate all the possible solutions and select one that minimizes the objective function. What is required is a computational scheme that selects, in an orderly fashion, a small subset of the possible solutions which converges to a minimum solution.

The simplex procedure, devised by Dantzig [5], is such a scheme. This procedure finds an extreme point and determines whether it is the minimum. If it is not, the procedure finds an adjacent extreme point whose corresponding value of the objective function is less than or equal to the preceding value. In a finite number of such steps (usually between m and $3m$), a minimum feasible solution is found. The simplex method makes it possible to discover whether the problem has no finite minimum solutions or no feasible solutions.

CHAPTER 2

THE SIMPLEX METHOD

2.1 Basic Method

To begin the simplex procedure, we arrange the problem matrix as shown in Tableau 2-1. There are many variations of the table. We have chosen to use the form as in [6].

From the original equations of the problem given by $\sum_{j=1}^{n} a_{ij}x_j = b_i$, $i = 1, 2, ..., m$, we have let $x_{i0} = b_i$ and $x_{ij} = a_{ij}$. The z_j for $j = 0, 1, ..., n$ are obtained by taking the inner product of the j^{th} vector with the column vector labeled **c**, that is,

$$z_0 = \sum_{i=1}^{m} c_i x_{i0} \tag{2.1}$$

Tableau 2-1. Initial step of simplex method.

				c_1	c_2	·	c_l	·	c_m	c_{m+1}	·	c_j	·	c_k	·	c_n
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	·	$\mathbf{P}_l$	·	$\mathbf{P}_m$	$\mathbf{P}_{m+1}$	·	$\mathbf{P}_j$	·	$\mathbf{P}_k$	·	$\mathbf{P}_n$
1	$\mathbf{P}_1$	c_1	x_{10}	1	0	·	0	·	0	$x_{1,m+1}$	·	x_{1j}	·	x_{1k}	·	x_{1n}
2	$\mathbf{P}_2$	c_2	x_{20}	0	1	·	0	·	0	$x_{2,m+1}$	·	x_{2j}	·	x_{2k}	·	x_{2n}
·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
l	$\mathbf{P}_l$	c_l	x_{l0}	0	0	·	1	·	0	$x_{l,m+1}$	·	x_{lj}	·	x_{lk}	·	x_{ln}
·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
m	$\mathbf{P}_m$	c_m	x_{m0}	0	0	·	0	·	1	$x_{m,m+1}$	·	x_{mj}	·	x_{mk}	·	x_{mn}
m+1	---	-	z_0	0	0	·	0	·	0	$z_{m+1}-c_{m+1}$	·	$z_j - c_j$	·	$z_k - c_k$	·	$z_n - c_n$

$$z_j = \sum_{i=1}^{m} c_i x_{ij} \, , \; j = 0, 1, \ldots, n \qquad (2.2)$$

The elements z_0 and z_j - c_j are entered in the $(m + 1)^{st}$ row of their respective columns. The z_j - c_j for those vectors in the basis will always equal zero. If all the numbers z_j - $c_j \leq 0$ for $j = 0, 1, \ldots, n$, then the solution $\mathbf{X}_0 = (x_{10}, x_{20}, \ldots, x_{m0}) = (b_1, b_2, \ldots, b_m)$ is a minimum feasible solution, and the corresponding value of the objective function is z_0. We will assume at least one $z_j - c_j \geq 0$ and compute a new feasible solution whose basis contains $m - 1$ vectors of the original basis $\mathbf{P}_1, \ldots, \mathbf{P}_m$. In searching for a new vector to enter the basis, we select the one which corresponds to the

$$\max_j (z_j - c_j)$$

If there are ties, we will select the vector with the lowest index j.

In our example, let

$$\max_k (z_k - c_k) = z_k - c_k > 0$$

The vector $\mathbf{P}_k$ is to be introduced into the basis. We next compute

$$\theta = \min_i \frac{x_{i0}}{x_{ik}} \text{ for } x_{ik} > 0$$

If all $x_{ik} \leq 0$, we can find a feasible solution whose value of the objective function can be made arbitrarily small. The computation is then complete. However, assume some $x_{ik} \geq 0$ and

$$\theta = \min_{i} \frac{x_{i0}}{x_{ik}} = \frac{x_{l0}}{x_{lk}}$$

Vector $\mathbf{P}_l$ will be the one removed from the basis. In order to obtain the new solution $\mathbf{X}'_0$, the new vectors $\mathbf{X}'_j$, and the corresponding $z'_j - c_j$, all elements in Tableau 2-1 for rows $i = 1, 2, \ldots, m+1$ and columns $j = 0, 1, \ldots, n$ are transformed by the formulas

$$x'_{ij} = x_{ij} - \frac{x_{lj}}{x_{lk}} x_{ik} \tag{2.3}$$

$$x_{lj} = \frac{x_{lj}}{x_{lk}} \tag{2.4}$$

where

$$z'_0 = x'_{m+1,0} \quad \text{and} \quad z'_j - c_j = x'_{m+1,j}$$

These general formulas (2.3) and (2.4) apply to all elements of the tableau including the $\mathbf{P}_0$ column and the $(m+1)^{st}$ row. The transformation defined by (2.3) and (2.4) is equivalent to the complete elimination formulas when the pivot element is x_{lk}.

Once an initial tableau has been constructed, the simplex method calls for the iteration of:

1. The testing of the $z_j - c_j$ elements to determine whether $z_j - c_j \leq 0$ for all j.
2. The selection of the vector to be introduced into the basis if some $z_j - c_j > 0$, i.e., the selection of the vector with maximum

$z_j - c_j$. The column associated with vector $\mathbf{P}_j$ is referred to as the *pivot column.*

3. The selection of the vector to be removed from the basis to ensure feasibility of the new solution, i.e., the vector with $\min\limits_{i} \frac{x_{i0}}{x_{ik}}$ for those $x_{ik} > 0$, where k corresponds to the vector selected in Step 2. The i^{th} row is referred to as the *pivot row.* If all $x_{ik} \leq 0$, then the solution is unbounded.
4. The transformation of the tableau by elementary row operations to obtain the new solution and associated elements.

Each such iteration produces a new basic feasible solution, and we eventually obtain a minimum solution or determine an unbounded solution. If ties occur in Steps 2 or 3, we use the lowest-index rule discussed earlier.

Example 2.1: As an example, we will solve the following LP problem by means of the simplex method:

$$\begin{array}{lllll} \text{minimize } z = x_1 - x_2 & & & & \\ \text{st} & 3x_1 + 2x_2 + x_3 & & & = 10 \\ & 3x_1 + 4x_2 & + x_4 & & = 20 \\ & x_1 & & + x_5 & = 3 \\ & x_j \geq 0, j = 1, \ldots, 5 & & & \end{array}$$

We begin by forming the initial tableau (Tableau 2-2). The initial basis consists of $\mathbf{P}_3$, $\mathbf{P}_4$, $\mathbf{P}_5$, and the corresponding solution is $\mathbf{X} = (x_3, x_4, x_5) = (10, 20, 3)$. Since $c_3 = c_4 = c_5 = 0$, the corresponding value of the objective function, z, equals zero. $\mathbf{P}_2$ is selected to enter the basis, since

$$\max_j (z_j - c_j) = z_2 - c_2 = 1 > 0$$

When we compute θ we have a tie in the 1st and 2nd rows. By the lowest-index rule, we choose row 1. Thus

$$\theta = \min \frac{x_{10}}{x_{12}} = 5$$

and so $\mathbf{P}_3$ is removed from the basis. The pivot element, 2, is denoted in Tableau 2-2 in bold. We transform the initial tableau to obtain Tableau 2-3 with a new solution $\mathbf{X}' = (x_2, x_4, x_5) = (5, 0, 3)$ and new objective function value of -5. Since $\max(z'_j - c_j) = 0$, this solution is a minimum feasible solution.

Tableau 2-2. Initial Step of Example 2.1.

				1	-1	0	0	0	
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	θ
1	$\mathbf{P}_3$	0	10	3	**2**	1	0	0	5
2	$\mathbf{P}_4$	0	20	3	4	0	1	0	5
3	$\mathbf{P}_5$	0	3	1	0	0	0	1	-
4	---	-	0	-1	1	0	0	0	

Tableau 2-3. Vector P_2 replaces vector P_3.

i	Basis	c	P_0	P_1	P_2	P_3	P_4	P_5
1	P_2	-1	5	$3/2$	1	$1/2$	0	0
2	P_4	0	0	-3	0	-2	1	0
3	P_5	0	3	1	0	0	0	1
4	---	-	-5	$-5/2$	0	$-1/2$	0	0

2.2 Addition of Slack Variables

The previous problem is an example of the simplest LP problem; that is, a problem in which the constraints are written as equalities. If, however, the problem had been written in the form $\mathbf{AX} \leq \mathbf{b}$, we would need to add additional variables, called *slack variables*, in order to have an initial basis. These slack variables represent the amount of unused resource and have associated cost coefficients set equal to zero.

Example 2.2: Consider the problem:

$$\begin{aligned} &\text{minimize } z = x_1 + x_2 - 2x_3 \\ &\text{st} \quad -x_1 \qquad\quad - 2x_3 \leq 5 \\ &\qquad\; 2x_1 - 3x_2 + x_3 \leq 3 \\ &\qquad\; 2x_1 + 5x_2 + 6x_3 \leq 5 \\ &\qquad\; x_j \geq 0 \quad j = 1, 2, 3 \end{aligned}$$

By adding the slack variables x_4, x_5, and x_6, the problem becomes

$$\begin{aligned} &\text{minimize } z = x_1 + x_2 - 2x_3 \\ &\text{st} \quad -x_1 \qquad\quad - 2x_3 + x_4 \qquad = 5 \\ &\qquad\; 2x_1 - 3x_2 + x_3 \qquad + x_5 = 3 \end{aligned}$$

$$2x_1 + 5x_2 + 6x_3 \qquad\qquad + x_6 = 5$$

$$x_j \geq 0 \quad j = 1, \ldots, 6$$

The constraint equations now contain a starting basis of the slack vectors ($\mathbf{P}_4$, $\mathbf{P}_5$, $\mathbf{P}_6$) with the associated first feasible solution of $x_1 = x_2 = x_3 = 0$, $x_4 = 5$, $x_5 = 3$, $x_6 = 5$. The corresponding value of the objective function is zero since all cost coefficients for the slack variables are zero.

Now we are able to continue the simplex method (Tableaus 2-4 and 2-5). We see that in Tableau 2-5 that the $\max(z_j - c_j) = 0$; i.e., that the minimum feasible solution has been reached. The solution is $\mathbf{X}'$ = (x_4, x_5, x_3) = $(20/3, 13/6, 5/6)$ and the objective function value is $-5/3$.

Tableau 2-4. Initial tableau of Example 2.2.

				1	1	-2	0	0	0	
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$	θ
1	$\mathbf{P}_4$	0	5	-1	0	-2	1	0	0	-
2	$\mathbf{P}_5$	0	3	2	-3	1	0	1	0	3
3	$\mathbf{P}_6$	0	5	2	5	**6**	0	0	1	$5/6$
4	---	-	0	-1	-1	2	0	0	0	

Tableau 2-5. Vector $\mathbf{P}_3$ replaces vector $\mathbf{P}_6$.

1	$\mathbf{P}_4$	0	$20/3$	$-1/3$	$5/3$	0	1	0	$1/3$
2	$\mathbf{P}_5$	0	$13/6$	$5/3$	$-23/6$	0	0	1	$-1/6$
3	$\mathbf{P}_3$	-2	$5/6$	$1/3$	$5/6$	1	0	0	$1/6$
4	---	-	$-5/3$	$-5/3$	$-8/3$	0	0	0	$-1/3$

2.3 Artificial Basis Technique

Other problems have constraints written in the form $\mathbf{AX} \geq \mathbf{b}$. For these problems, the artificial basis technique is used to start the simplex method. This procedure also determines whether or not the problem has any feasible solutions. This technique is also referred to as the Big-M Method (Winston [18]).

For the artificial basis technique we augment the general LP system (Equations (1.1) – (1.3)) as follows:

$$\begin{array}{llcccl}
\text{minimize} & \multicolumn{5}{l}{c_1x_1 + \cdots + c_nx_n + wx_{n+1} + wx_{n+2} + \cdots + wx_{n+m}} \\
\text{st} & a_{11}x_1 + \cdots + a_{1n}x_n + x_{n+1} & & & & = b_1 \\
 & a_{21}x_1 + \cdots + a_{2n}x_n & & + x_{n+2} & & = b_2 \\
 & \quad \vdots & & \vdots & & \vdots \\
 & a_{m1}x_1 + \cdots + a_{mn}x_n & & & +x_{m+n} & = b_m \\
 & \multicolumn{5}{l}{x_j \geq 0 \text{ for } j = 1, \ldots, n, n+1, \ldots, n+m}
\end{array}$$

The quantity w is taken to be an unspecified large positive number. The vectors $\mathbf{P}_{n+1}, \mathbf{P}_{n+2}, \ldots, \mathbf{P}_{n+m}$ form an artificial basis for the augmented system. If there is at least one feasible solution to the original problem, then this solution is also a feasible solution for the augmented system. The simplex method will then obtain the minimum solution, in which it is impossible for one of the artificial variables, x_{n+i}, to appear with a positive value. If the minimum feasible solution for the augmented system contains at least one $x_{n+i} > 0$, then the original problem is not feasible.

Each $z_j - c_j$ will have a w coefficient and a non - w coefficient which are independent of each other. The associated computational procedure is set up as Tableau 2-6. For each j, the non - w component and the w component of $z_j - c_j$ have been placed in the $(m+1)^{st}$ and $(m+2)^{nd}$ rows, respectively, of that column.

This tableau is treated like the original simplex tableau (Tableau 2-1) except that the vector introduced into the basis is associated with the largest positive element in the $(m+2)^{nd}$ row. For the first iteration, the vector corresponding to $\max_j \sum_{i=1}^{n} x_{ij}$ is introduced into the basis. The elements in the $(m+2)^{nd}$ row are also transformed by the usual elimination procedure. Once an artificial vector is removed from the basis, it is never selected to reenter the basis. Hence we do not have to transform the last m columns of the tableau. Even if there are artificial vectors in the basis, the iteration may not remove one of them. At least m iterations are required to exchange them with the columns of the given problem when a full artificial basis is used.

Tableau 2-6. Initial tableau for artificial basis technique.

				c_1	c_2	·	c_k	·	c_n	w	·	w	·	w
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	·	$\mathbf{P}_k$	·	$\mathbf{P}_n$	$\mathbf{P}_{n+1}$	·	$\mathbf{P}_{n+l}$	·	$\mathbf{P}_{n+m}$
1	$\mathbf{P}_{n+1}$	w	$x_{n+1,0}$	x_{11}	x_{12}	·	x_{1k}	·	x_{1n}	1	·	0	·	0
2	$\mathbf{P}_{n+2}$	w	$x_{n+2,0}$	x_{21}	x_{22}	·	x_{2k}	·	x_{2n}	0	·	0	·	0
·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
l	$\mathbf{P}_{n+l}$	w	$x_{n+l,0}$	x_{l1}	x_{l2}	·	x_{lk}	·	x_{ln}	0	·	1	·	0
·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
m	$\mathbf{P}_{n+m}$	w	$x_{n+m,0}$	x_{m1}	x_{m2}	·	x_{mk}	·	x_{mn}	0	·	0	·	1
$m+1$	---	-	0	$-c_1$	$-c_2$	·	$-c_k$	·	$-c_n$	0	·	0	·	0
$m+2$	---	-	$\sum x_{n+i,0}$	$\sum x_{i1}$	$\sum x_{i2}$	·	$\sum x_{ik}$	·	$\sum x_{in}$	0	·	0	·	0

We continue to select a vector to be introduced into the basis using the element in the $(m+2)^{nd}$ row as criterion, until either (1) all the artificial vectors are removed from the basis or (2) no positive $(m+2)^{nd}$ element exists. The first alternative implies that all the elements in the $(m+2)^{nd}$ row equal zero and the corresponding basis is a feasible basis for the original problem. We then apply the regular simplex method to determine the minimum feasible solution. In the second alternative, if the $(m+2, 0)$ element, i.e., the artificial part of the corresponding value of the objective function, is greater than zero, then the original problem is not feasible. If the $(m+2, 0)$ element is equal to zero, then we have a degenerate feasible solution to the original problem which contains at least one artificial vector. The artificial variables have values of zero. However, the minimum feasible solution has not been reached. We continue the iterations by introducing a vector that corresponds to the maximum positive element in the $(m+1)^{st}$ row which is above a zero element in the $(m+2)^{nd}$ row. This criterion is used until there are no more positive $(m+1)^{st}$ elements over a zero in the $(m+2)^{nd}$ row. The final solution may or may not contain artificial variables with values equal to zero. For both alternatives (1) and (2), all the $(m+2, j)$ elements are less than or equal to zero, with the possible exception of the $(m+2, 0)$ element. The latter element is always nonnegative, and its value is nonincreasing.

Whenever the original problem contains some unit vectors, these vectors along with the necessary artificial ones should be used for the initial basis. Doing this will decrease the total number of iterations.

Example 2.3: Consider the problem:

$$\begin{aligned} &\text{minimize } z = 4x_1 + 4x_2 + x_3 \\ &\text{st} \quad x_1 + x_2 + x_3 \leq 2 \\ &\quad\quad 2x_1 + x_2 \leq 3 \\ &\quad\quad 2x_1 + x_2 + 3x_3 \geq 3 \\ &\quad\quad x_j \geq 0 \text{ for } j = 1, 2, 3 \end{aligned}$$

We add slack variables x_4 and x_5 to constraints 1 and 2, respectively, and subtract slack variable x_6 from constraint 3 to make each of the constraints equalities. In order to obtain a basis we must add artificial variable x_7 to constraint 3. We must also add wx_7 to the objective function. The problem becomes

$$\text{minimize } z = 4x_1 + 4x_2 + x_3 + wx_7$$

$$\begin{array}{llr} \text{st} & x_1 + x_2 + x_3 + x_4 & = 2 \\ & 2x_1 + x_2 + x_5 & = 3 \\ & 2x_1 + x_2 + 3x_3 - x_6 + x_7 & = 3 \\ & x_j \geq 0 \text{ for } j = 1, \ldots, 7 & \end{array}$$

The initial tableau for Example 2.3 is recorded in Tableau 2-7. The **P**$_3$ column has the largest positive element in its $(m+2)^{nd}$ row, so vector **P**$_3$ is chosen to enter the basis. Calculating θ tells us that vector **P**$_7$ is removed from the basis. The results of performing the simplex elimination calculations are recorded in Tableau 2-8.

Tableau 2-7. Initial Step of Example 2.3.

				4	4	1	0	0	0	w	
i	Basis	c	**P**$_0$	**P**$_1$	**P**$_2$	**P**$_3$	**P**$_4$	**P**$_5$	**P**$_6$	**P**$_7$	θ
1	**P**$_4$	0	2	1	1	1	1	0	0	0	2
2	**P**$_5$	0	3	2	1	0	0	1	0	0	-
3	**P**$_7$	w	3	2	1	**3**	0	0	-1	1	1
4	---	-	0	-4	-4	-1	0	0	0	0	
5	---	-	3	2	1	3	0	0	-1	0	

Tableau 2-8. Vector P$_3$ replaces vector P$_7$.

1	**P**$_4$	0	1	$\frac{1}{3}$	$\frac{2}{3}$	0	1	0	$\frac{1}{3}$
2	**P**$_5$	0	3	2	1	0	0	1	0
3	**P**$_3$	1	1	$\frac{2}{3}$	$\frac{1}{3}$	1	0	0	$-\frac{1}{3}$
4			1	$-\frac{10}{3}$	$-\frac{11}{3}$	0	0	0	$-\frac{1}{3}$
5			0	0	0	0	0	0	0

The $(m+2)^{nd}$ row consists of all zeroes and all elements in the $(m+1)^{st}$ row are less than or equal to zero, so we have reached the optimal solution of $z = 1$, $x_1 = x_2 = 0$, $x_3 = 1$.

CHAPTER 3

DUAL SIMPLEX METHOD

3.1 Introduction

Associated with every LP problem as defined in Chapter 1 is a corresponding problem called the *dual problem*. The original problem is called the *primal*. An interesting historical note is the introduction of the term primal around 1954 by George Dantzig. He had been told he needed a word to stand for "the original problem of which this is the dual." He asked his father, Tobias Dantzig, mathematician and author, well-known for his books popularizing the history of mathematics. His father knew both Latin and Greek and suggested primal as the antonym since both primal and dual derive from the Latin. That was his father's one and only contribution to linear programming [5].

The optimal solution of either problem reveals information about the optimal solution of the other. If the initial simplex tableau for the primal problem contains an $m \times m$ unit matrix, then the solution of either problem by the simplex procedure yields an explicit solution to the other. Because we considered the standard LP problem to be a minimization problem, we will now designate the minimization problem to be the primal problem.

Recall that in the primal problem we want to find a column vector $\mathbf{X} = (x_1\ x_2, \ldots, x_n)$ which minimizes the linear functional

$$f(\mathbf{X}) = \mathbf{cX} \qquad (3.1)$$

subject to the conditions

$$\mathbf{AX} = \mathbf{b} \qquad (3.2)$$

and

$$\mathbf{X} \geq \mathbf{0} \qquad (3.3)$$

The dual problem is to find a row vector $\mathbf{W} = (w_1, w_2, \ldots, w_m)$ which maximizes the linear functional

$$g(\mathbf{W}) = \mathbf{Wb} \qquad (3.4)$$

subject to the conditions

$$\mathbf{WA} \leq \mathbf{c} \qquad (3.5)$$

In the dual problem, the variables W_i are not restricted to be nonnegative. Multiplying the $1 \times m$ row vector $\mathbf{W}$ by the $m \times n$ matrix $\mathbf{A}$, we have the following explicit representation of (3.5):

$$a_{11}w_1 + a_{12}w_2 + \cdots + a_{m1}w_m \leq c_1$$

$$a_{12}w_1 + a_{22}w_2 + \cdots + a_{m2}w_m \leq c_2$$

$$\cdot\cdot \qquad \cdot\cdot \qquad \cdots \qquad \cdot\cdot \qquad \cdot$$

$$a_{1n}w_1 + a_{2n}w_2 + \cdots + a_{mn}w_m \leq c_n$$

The matrix of coefficients for the above inequalities is given by $\mathbf{A}^T$, the transpose of the matrix $\mathbf{A}$. Note that there is a dual variable for each equation of (3.2).

Associated with the primal and dual problems is the following theorem:

Theorem 3.1: The Duality Theorem. If either the primal or the dual problem has a finite optimal solution, then the other problem has a finite optimal solution and min f(**X**) = max g(**W**). If either problem has an unbounded optimal solution, then the other problem has no feasible solutions. [Note that for the dual problem, a solution is feasible if it satisfies (3.5). The variables are not required to be nonnegative.]

Dantzig [4] states Theorem 3.1 in a slightly different form.

Theorem 3.1a. If feasible solutions to both the primal and dual systems exist, there is an optimal solution to both systems and min f(**X**) = max g(**W**).

3.2 The Dual Simplex Method

The dual simplex algorithm is the work of C. E. Lemke. It apparently originated in the course of applying the primal simplex method to the dual of an LP problem. While doing this, Lemke realized that an algorithm that was different in the sense of feasible points leading to optimality could be devised, and that it could be applied to either primal or dual problems (Cooper and Steinberg [3]).

To begin the dual simplex procedure we will assume that we have selected a basis $\mathbf{B} = (\mathbf{P}_1\mathbf{P}_2...\mathbf{P}_m)$ such that at least one element of $\mathbf{B}^{-1}\mathbf{b}$ is negative and $\mathbf{c}^0\mathbf{B}^{-1}\mathbf{P}_j \leq c_j$ for all j, where $\mathbf{c}^0 = (c_1, c_2, ..., c_m)$ is a row vector. A solution to the dual constraints is given by $\mathbf{W}^0 = \mathbf{c}^0\mathbf{B}^{-1}$ with its corresponding value of the objective function being $\mathbf{c}^0\mathbf{B}^{-1}\mathbf{b}$.

We need a computational procedure for the dual which will yield a maximizing solution, and hence, by the duality theorem, a minimizing solution to the primal. This procedure must then determine a new basis for which

1. The dual inequalities will still be satisfied.
2. The value of the dual objective function will increase (or remain the same if the dual solution is degenerate) until the maximum or unbounded solution is reached.

Proceeding in this way will always preserve optimality of the primal, and in a finite number of steps, determine a feasible and optimal solution to the primal.

We will denote the rows of $\mathbf{B}^{-1}$ by $\mathbf{B}_i$. The m components of the nonfeasible solution of the primal are given by $x_{i0} = \mathbf{B}_i\mathbf{b}$ for $i = 1, 2, ..., m$. Let $x_{l0} = \mathbf{B}_l\mathbf{b} = \min\limits_{i} \mathbf{B}_i\mathbf{b} < 0$. The vector $\mathbf{P}_l$ will be the one removed from the basis. For those vectors not in the basis, i.e., for those having $\mathbf{W}^0\mathbf{P}_j < c_j$, compute $x_{lj} = \mathbf{B}_l\mathbf{P}_j$. We will assume that at least one $x_{lj} < 0$. For the set of $x_{lj} < 0$ form the ratios $\frac{z_j - c_j}{x_{lj}}$. Let

$$\theta = \min_{x_{lj}<0} \frac{z_j - c_j}{x_{lj}} = \frac{z_k - c_k}{x_{lk}} > 0 \tag{3.6}$$

The vector $\mathbf{P}_k$ is selected to replace $\mathbf{P}_l$, and the new basis will yield a solution to the dual constraints. The new basis is $\overline{\mathbf{B}} =$ $(\mathbf{P}_1 \ldots \mathbf{P}_l \mathbf{P}_k \mathbf{P}_{l+1} \ldots \mathbf{P}_m)$ and $\overline{\mathbf{B}}^{-1}$ is obtained by application of the following formulas on $\mathbf{B}^{-1}$:

Let b_{ij} denote the element in the i^{th} row and j^{th} column of $\mathbf{B}^{-1}$ and let $\overline{b_{ij}}$ be the corresponding element of $\overline{\mathbf{B}}^{-1}$. Then $\overline{b_{ij}}$ are given by

$$\overline{b_{ij}} = b_{ij} - \frac{b_{lj}}{x_{lk}} x_{ik} \text{ for } i \neq l$$

$$\overline{b_{lj}} = \frac{b_{lj}}{x_{lk}}$$

The new solution to the dual constraints is

$$\overline{\mathbf{W}} = \mathbf{W}^o - \theta \mathbf{B}_l \tag{3.7}$$

and the corresponding value of the objective function is

$$\overline{\mathbf{W}}\mathbf{b} = \mathbf{W}^o\mathbf{b} - \theta x_{l0} \tag{3.8}$$

The new solution to the primal constraints can be computed by the usual elimination formulas or directly from $\overline{\mathbf{X}} = \mathbf{B}^{-1}\mathbf{b}$. If all $\overline{\mathbf{X}} \geq 0$, then we have determined an optimal feasible solution to the primal. If not, then we know we have made at least $\overline{x}_{l0} = \overline{\mathbf{B}}\mathbf{b} > 0$. For this situation, we repeat the above dual simplex process of selecting the vector to be removed and then the vector to be introduced into the basis, until we find a basis that solves the dual and is also an admissible basis for the primal or until we have determined that the dual has an unbounded solution and hence that there are no feasible solutions to the primal. The latter case occurs when, in computing the $x_{lj} = \mathbf{B}_{lj}\mathbf{P}_j$, all $x_{lj} > 0$. If this is true, we have from (3.6) and (3.7) that we can construct a solution to the dual constraints for any $\theta > 0$, since

$$\mathbf{W}^o\mathbf{P}_j - \theta\mathbf{B}_l\,\mathbf{P}_j = \mathbf{W}^o\mathbf{P}_j - \theta x_{lj} \leq \mathbf{W}^o\mathbf{P}_j < c_j$$

From (3.8) the corresponding value of the objective function can be made as large as possible, since $x_{l0} < 0$. The dual simplex procedure can be employed as a variation of the original simplex procedure since it uses the information contained in its tableau.

Example 3.1: Consider the problem:

$$\begin{aligned}
&\text{minimize } z = 2x_1 + x_2 + 3x_3\\
&\text{st} \quad -x_1 + 2x_2 - x_3 + x_4 = -4\\
&\qquad 2x_1 + x_2 + x_3 + x_5 = 8\\
&\qquad -x_1 + x_3 + x_6 = 0\\
&\qquad x_j \geq 0, j = 1, \dots, 6
\end{aligned}$$

The dual of this problem is to

$$\text{maximize } z = -4w_1 + 8w_2$$

$$\begin{array}{lrl} \text{st} & -w_1 + 2w_2 - w_3 & \le 2 \\ & 2w_1 + \ w_2 + \quad & \le 1 \\ & -w_1 + \ w_2 + w_3 & \le 3 \\ & w_1 \qquad\qquad & \le 0 \\ & w_2 \qquad & \le 0 \end{array}$$

An initial basis $\mathbf{B} = (\mathbf{P}_4\mathbf{P}_5\mathbf{P}_6) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Setting up the primal in the usual simplex tableau, we have Tableau 3-1.

Since all $z_j - c_j$ elements are nonpositive, the basis **B** is a feasible solution for the dual, i.e., optimal but not feasible for the primal. The dual solution is

$$\mathbf{W}^0 = c^0\mathbf{B}^{-1} = (0 \quad 0 \quad 0)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (0 \quad 0 \quad 0)$$

Applying the dual algorithm, we see that $\mathbf{P}_4$ is to be eliminated ($x_l = x_4 = -4$), and vector $\mathbf{P}_1$ is to be introduced into the basis because

Tableau 3-1. Initial Step of Example 3.1.

				2	1	3	0	0	0
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$
1	$\mathbf{P}_4$	0	-4	**-1**	2	-1	1	0	0
2	$\mathbf{P}_5$	0	8	2	1	1	0	1	0
3	$\mathbf{P}_6$	0	0	-1	0	1	0	0	1
4	---	-	0	-2	-1	-3	0	0	0

Tableau 3-2. Vector P_1 replaces vector P_4.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$
1	$\mathbf{P}_1$	2	4	1	-2	1	-1	0	0
2	$\mathbf{P}_5$	0	0	0	5	-1	2	1	0
3	$\mathbf{P}_6$	0	4	0	-2	2	-1	0	1
4	---	-	8	0	-5	-1	-2	0	0

$$\theta = \frac{z_j - c_j}{z_{41}} = 2$$

The new tableau is as in Tableau 3-2.

Since all $x_i \geq 0$ and since $z_j - c_j \leq 0$, we have determined the optimal feasible solution. The primal optimal solution is $x_1 = 4$, $x_2 = x_3 = 0$, and that for the dual is $w_1 = -2$, $w_2 = 0$, with a common optimal value of the objective function of 8.

CHAPTER 4

INTEGER PROGRAMMING

4.1 Introduction

Recall from Chapter 1 that a linear programming problem in which some or all of the variables must be nonnegative integers is called an *integer programming* problem (IP). An IP in which all variables are required to be integers is called a *pure IP* problem. An IP in which only some of the variables are required to be integers is called a *mixed IP* problem. An IP in which all the variables must equal zero or one is called a *0-1 IP* problem or a 0-1 programming problem. The standard form of the IP is similar to that of an LP:

$$\begin{aligned}
\text{minimize } & c_1x_1 + c_2x_2 + \cdots + c_jx_j + \cdots + c_nx_n \\
\text{st} \quad & a_{11}x_1 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n = b_1 \\
& \cdot \qquad \cdot \qquad \cdot \qquad \cdot \\
& a_{i1}x_1 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n = b_i \\
& \cdot \qquad \cdot \qquad \cdot \qquad \cdot \\
& a_{m1}x_1 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n = b_m \\
& x_j \geq 0 \\
& x_p \text{ integer} \\
& x_q = 0 \text{ or } 1
\end{aligned}$$

where x_p and x_q represent those variables that are required to be integer or 0-1 variables, respectively.

4.2 Summary of Solution Methods

Since the feasible region for any IP is contained in the feasible region for the corresponding LP relaxation (the LP obtained by omitting all integer or 0-1 constraints), one might initially assume that an optimal solution for an IP can be obtained by first solving the LP relaxation and then rounding off (to the nearest integer) each variable that is required to be an integer and that assumes a fractional value in the optimal solution to the LP relaxation. To see that this method is false, consider the following example:

Example 4.1: maximize $z = 21x_1 + 11x_2$

$$\begin{aligned} \text{st} \quad & 7x_1 + 4x_2 \leq 13 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \text{ integer} \end{aligned} \tag{4.1}$$

Applying the above method, we first find the optimal solution to the LP relaxation: $x_1 = {}^{13}\!/_{7}$, $x_2 = 0$. Rounding this solution yields the solution $x_1 = 2$, $x_2 = 0$ as a possible optimal solution to (4.1). We can see in Figure 4-1 that $x_1 = 2$, $x_2 = 0$ is infeasible for (4.1) so it cannot possibly be the optimal solution to (4.1). Even if we round x_1 downward (yielding

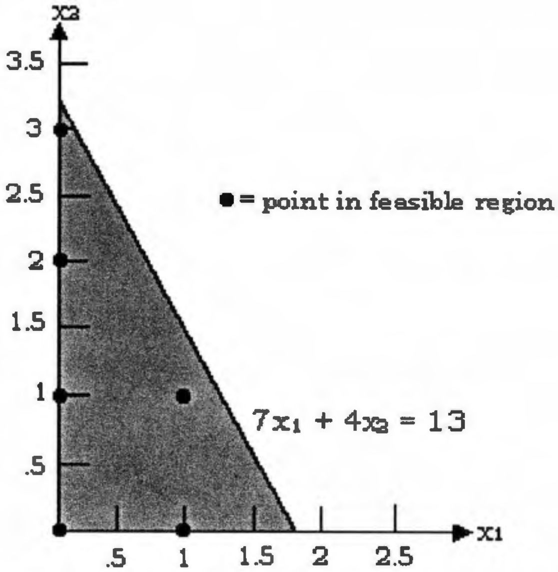


Figure 4-1. Feasible region for Example 4.1.

the candidate solution $x_1 = 1$, $x_2 = 0$), we do not obtain the optimal solution ($x_1 = 0$, $x_2 = 3$) [18].

Since rounding does not yield the optimal integer solution in most cases, other solution methods are needed. The procedures for solving IP problems that will be discussed in Section II can be grouped into two computational categories: cutting plane and enumeration. The basic idea of the cutting plane method is as follows: Given an IP problem, solve the corresponding LP problem. If the optimal solution contains only integer-valued variables, then this optimal solution is also optimal for the IP. If, however, some variables in the optimal LP solution are not integer-valued, then a new constraint is added to the problem, and the new LP problem is solved. Again, the optimal solution to the new LP problem is checked to see if it meets the integer requirements. If so, it is the optimal solution to the original IP problem. If the integer requirements have not been met then another new constraint is added, and the process continues until one of the corresponding LP problems has an all integer optimal solution.

Enumerative methods, used to solve both pure and mixed-integer problems, include additive algorithms and branch-and-bound methods. Additive algorithms are designed to solve 0-1 programming problems with n binary variables and 2^n possible solutions. The procedure does not use simplex techniques but substitutes new 0-1 solutions based on systematic rules for improving a solution which rely on information implied by the original problem constraints. It is an additive procedure in that only additions and subtractions are used in the computation. Branch-and-bound methods use information obtained from successive solutions to related continuous problems to generate new problems with bounds on selected variables and the objective function in such a way as to restrict the total number of continuous problems which need to be solved [6].

To illustrate the differences in approach to solving an IP problem, consider the following example:

Example 4.2: $\max z = 8x_1 + 5x_2$

$$\begin{aligned} \text{st} \quad x_1 + x_2 &\leq 6 \\ 9x_1 + 5x_2 &\leq 45 \\ x_1, x_2 &\geq 0 \text{ and integer} \end{aligned}$$

Figure 4-2 shows the feasible region for the LP relaxation of Example 4.2. Included in the figure is a cutting constraint, $3x_1 + 2x_2 = 15$. Notice that this constraint "cuts off" the portion of the feasible region which contained the optimal LP solution. It is possible that more cutting planes would need to be added in order to find the optimal IP solution.

Figure 4-3 illustrates the branch-and-bound method with its "divide and conquer" approach. In this figure, the feasible region has been divided into three subregions by the constraints $x_1 = 3$ and $x_1 = 4$. The two regions that will continue to be searched have been labeled Subproblem 2 and Subproblem 1. The regions of these subproblems may need to be further subdivided until the optimal IP solution is found.

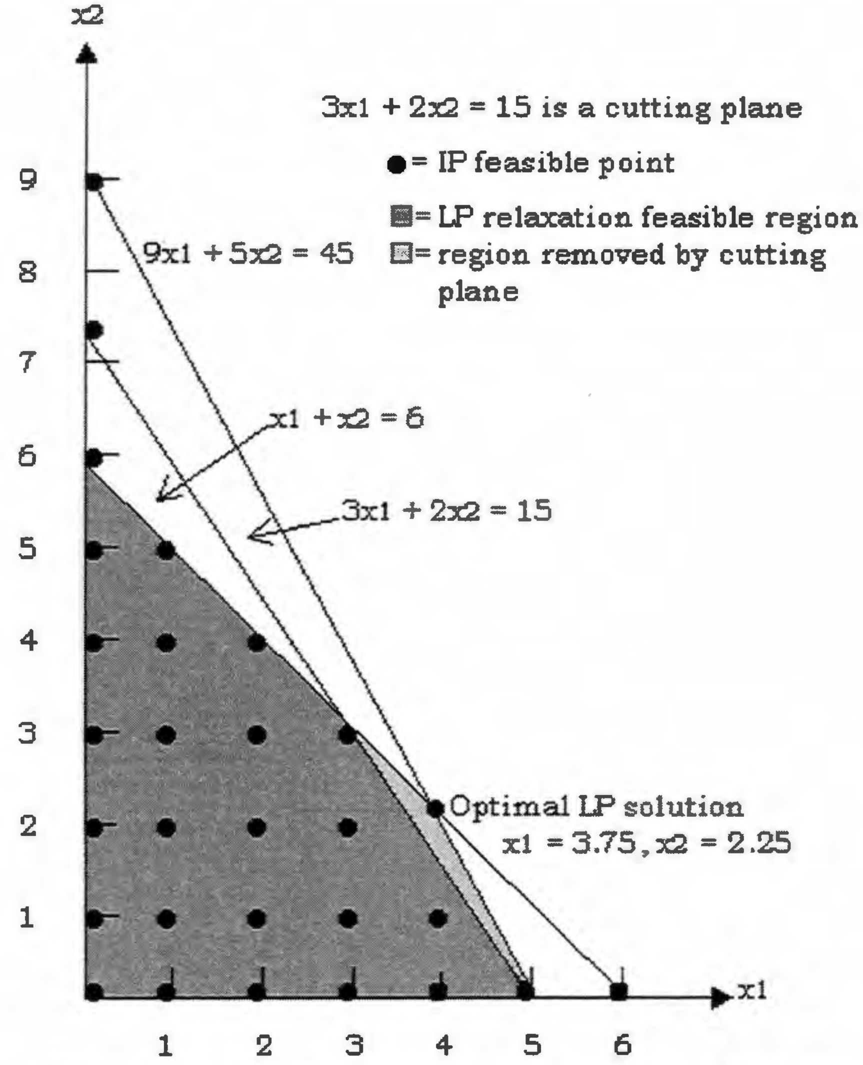


Figure 4-2. Graphical example of cutting plane method.

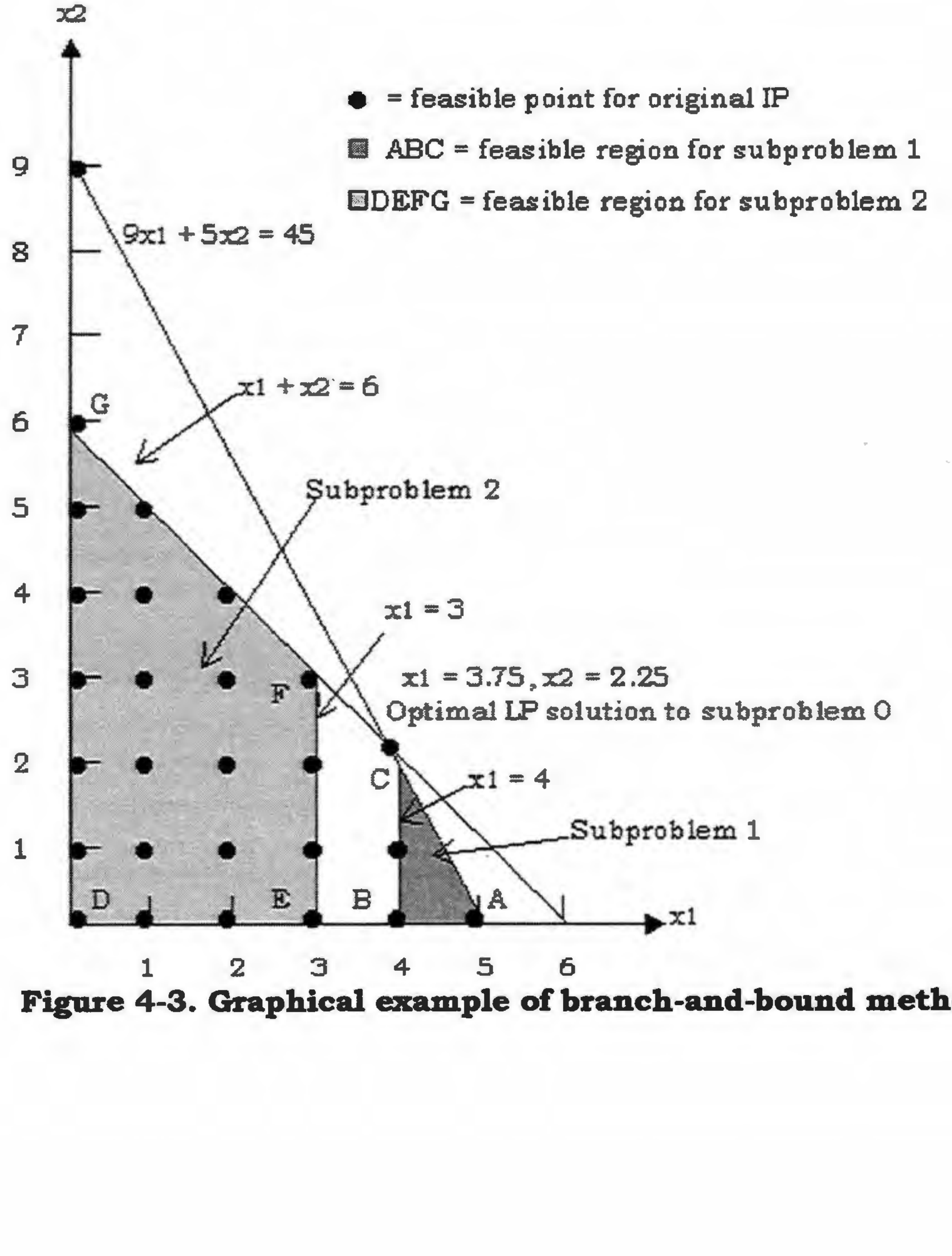

x2
x1
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
● = feasible point for original IP
▨ ABC = feasible region for subproblem 1
▨DEFG = feasible region for subproblem 2
9x1 + 5x2 = 45
x1 + x2 = 6
Subproblem 2
x1 = 3
x1 = 3.75, x2 = 2.25
Optimal LP solution to subproblem 0
x1 = 4
Subproblem 1
G
F
C
D
E
B
A

Figure 4-3. Graphical example of branch-and-bound method.

SECTION II: METHODS

CHAPTER 5

CUTTING PLANE METHOD

5.1 Introduction

Cutting plane algorithms have been used to solve many different IP problems, including the traveling salesman problem (TSP), the linear ordering problem, maximum cut problems, and packing problems. They have been proven to be useful computationally in the last few years, especially when combined with a branch-and-bound algorithm in a branch-and-cut framework. The branch-and-bound algorithm will be discussed in the next chapter and the branch-and-cut method in Chapter 8.

Some useful families of general inequalities have been developed for general problems. These include Gomory cutting planes, cuts based on knapsack problems, lift and project cutting planes, and Fenchel cutting planes. These inequalities, which are added to the problem, are called cutting planes because they are hyperplanes that cut off a portion of the convex set of feasible solutions for the related LP problem. Bard

[3] calls these cutting planes "violated cuts." We will concentrate on the Gomory cuts, which will be used again in Chapter 8.

The new constraints added at each step must have the following properties in order that the method to be valid:

1. Every feasible integer solution to the original problem must also be a feasible solution to the new problem after the addition of the constraint. In other words, addition of the new constraint does not eliminate any of the integer solutions from the set of feasible solutions. This ensures that the optimal integer solution will not be made infeasible.
2. The optimal solution to the LP problem solved at each step must become infeasible after the new constraint is added. Without this property, the optimal LP solution would also be optimal for the new LP problem, and nothing would have been gained by adding the constraint [3].

5.2 Gomory All-Integer Method

An early approach to the application of the cutting plane method required solving the LP relaxation by the basic simplex algorithm. If the optimal solution is not integer, then a cutting plane is introduced which preserves the optimality state of the primal problem but not the feasibility conditions. A single application of the dual algorithm restores feasibility and yields an optimal answer to the reduced problem. If this solution is integer-valued, the process stops. If not, a new constraint is introduced and the algorithm is repeated until an integer solution is found or until an indication is given that no integer solution exists for the original problem. This method has been superceded by a more efficient procedure, in which the problem, assumed to be given in integers, is transformed by a modified simplex algorithm that preserves the integer-valued characteristic of the complete tableau for all iterations [6].

We assume for starting conditions that the simplex tableau is all-integer-valued and that we start with a feasible solution for the dual simplex algorithm but not for the primal. Letting $\mathbf{P}_1, \ldots, \mathbf{P}_m$ form a feasible basis for the dual problem, the initial Tableau 5-1 is given for the minimization problem.

In Tableau 5-1 x_{00} is the value of the objective function; some $x_{i0} < 0$ for $i \geq 1$ (solution is not primal feasible); $x_{0j} = z_j - c_j \leq 0$ for $j \geq 1$ (solution is dual feasible); and an additional set of n-m constraints, $x'_j - x_j = 0$ for $j = m+1, \ldots, n$ with $x'_j \geq 0$ and $c'_j = c_j$, has been added to the tableau. With this set of constraints, we are now forced to determine a basic feasible solution that contains n nonnegative variables instead of the usual m; hence we are given the means by which an interior point of the original convex set can be expressed. However, we still lack the means to force us into the interior and toward the optimal integer-valued point. This is accomplished by a proper selection of cutting planes.

We wish to introduce new variables into the solution in a way that will enable us to preserve the integer characteristic of the tableau and to move toward the optimal integer solution that is in the convex set defined by the equations of Tableau 5-1. This will be accomplished by determining cutting hyperplanes defined as equalities in terms of the nonbasic variables and a new slack variable. Eliminating the new slack variable from the basis with a pivot element of −1 will preserve the integer tableau and move the solution point closer to the optimal integer point.

Tableau 5-1. Initial tableau with constraint row and column added.

				c_1	c_2	.	c_l	.	c_m	c_{m+1}	.	c_n	c_{m+1}	.	c_n	
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	.	$\mathbf{P}_l$	.	$\mathbf{P}_m$	$\mathbf{P}_{m+1}$	.	$\mathbf{P}_n$	$\mathbf{P}'_{m+1}$	.	$\mathbf{P}'_n$	$\mathbf{P}_{n+s}$
0	---	-	x_{00}	0	0	.	0	.	0	$x_{0,m+1}$	.	x_{0n}	0	.	0	
1	$\mathbf{P}_1$	c_1	x_{10}	1	0	.	0	.	0	$x_{1,m+1}$	.	x_{1n}	0	.	0	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
l	$\mathbf{P}_l$	c_l	x_{l0}	0	0	.	1	.	0	$x_{l,m+1}$	.	x_{ln}	0	.	0	
m	$\mathbf{P}_m$	c_m	x_{m0}	0	0	.	0	.	1	$x_{m,m+1}$	.	x_{mn}	0	.	0	
m+1	$\mathbf{P}'_{m+1}$	c_{m+1}	0	0	0	.	0	.	0	-1	.	0	1	.	0	
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	
n	$\mathbf{P}'_n$	c_n	0	0	0	.	0	.	0	0	.	-1	0	.	1	
n+1																

A finite number of applications of these new constraints leads us to the optimal integer solution.

To determine the form of the cutting constraint, consider any equation from Tableau 5-1 that has a corresponding $x_{l0} < 0$, for example, the l^{th} equation:

$$x_{l0} = x_l + x_{l,m+1}x_{m+1} + \cdots + x_{ln}x_n \tag{5.1}$$

where $x_{m+1}, \ldots, x_n$ are nonbasic variables.

We next rewrite each coefficient of (5.1) as a multiple of an integer and a remainder, i.e., in the form $b_{lj}\lambda + r_{lj}$, where b_{lj} is an integer, r_{lj} is a remainder, and λ is an unspecified positive number to be determined. The coefficients can then be expressed as

$$\begin{gathered} x_{lj} = b_{lj}\lambda + r_{lj} = \left\lfloor \frac{x_{lj}}{\lambda} \right\rfloor \lambda + r_{lj} \text{ for all } j \\ 1 = \left\lfloor \frac{1}{\lambda} \right\rfloor \lambda + r \\ 0 \le r_{lj} < \lambda \qquad 0 \le r < \lambda \qquad 0 < \lambda \end{gathered} \tag{5.2}$$

where $\lfloor x \rfloor$ indicates "greatest integer $\le$ x." If $\frac{x_{lj}}{\lambda} < 0$, then $\left\lfloor \frac{x_{lj}}{\lambda} \right\rfloor = b_{lj} < 0$ such that $b_{lj}\lambda + r_{lj} = x_{lj}$. Note that for $\lambda > 1$, we have $\left\lfloor \frac{1}{\lambda} \right\rfloor = 0$.

Substituting (5.2) into (5.1) and gathering appropriate terms, we have

$$r_{l0} + \lambda\left\{\left\lfloor \frac{x_{l0}}{\lambda} \right\rfloor - \left\lfloor \frac{x_{l,m+1}}{\lambda} \right\rfloor x_{m+1} - \ldots \left\lfloor \frac{x_{ln}}{\lambda} \right\rfloor x_n - \left\lfloor \frac{1}{\lambda} \right\rfloor x_l \right\} \qquad (5.3)$$

$$= r_{l,m+1}x_{m+1} + \cdots + r_{ln}x_n + rx_l$$

Note that any nonnegative integer values of the variables that satisfy Equation (5.1) will also satisfy (5.3). Such a substitution will make the right-hand side of (5.3) a nonnegative number.

The terms in braces can be rewritten as

$$\left\lfloor \frac{x_{l0}}{\lambda} \right\rfloor - \sum_{j \notin B} \left\lfloor \frac{x_{lj}}{\lambda} \right\rfloor x_j - \left\lfloor \frac{1}{\lambda} \right\rfloor x_l \qquad (5.4)$$

Note that the expression (5.4) must be a nonnegative integer.

For λ > 1, the last term of (5.4) is zero and by (5.2) and the above discussion we can rewrite (5.3) as

$$b_{l0} - \sum_{j \notin B} b_{lj}x_j \geq 0 \qquad (5.4a)$$

Thus (5.4a) is a cutting plane in terms of the current nonbasic variables, and the left-hand side of (5.4a) is a nonnegative integer. We transform (5.4a) into an equation by subtracting out the nonnegative slack variables x_{n+s}, i.e.,

$$b_{l0} - \sum_{j \notin B} b_{lj}x_j - x_{n+s} = 0 \qquad (5.5)$$

$$\text{or } b_{l0} = b_{l,m+1}x_{m+1} + b_{l,m+2}x_{m+2} + \cdots + b_{ln}x_n + x_{n+s}$$

Variable x_{n+s} is also an integer variable, as it is equal to the left-hand side of (5.4a). For any λ > 1, (5.5) is a constraint that must be satisfied by any integer solution to the original LP problem. The nonnegative integer variable x_{n+s} is introduced as a new variable into the problem. For the present solution, since all the nonbasic variables $x_{m+1} = \cdots = x_n = 0$, Equation (5.5) is infeasible as $b_{t0} < 0$. Thus, after a suitable selection of $\lambda > 1$, i.e., a value of λ which makes some element b_{tj} ($j \notin \mathbf{B}$) of (5.5) equal to -1. Equation (5.5) can be used as a cutting constraint.

We note that $b_{t0} < 0$ since it was assumed that $x_{t0} < 0$. Also, since we are applying the dual simplex method, some x_{tj} for j not in the basis must be negative or the problem is not feasible. By selecting λ large enough, all $\left\lfloor \frac{x_{tj}}{\lambda} \right\rfloor$ for $x_{tj} < 0$ yields a $b_{tj} = -1$; hence a pivot element of -1 is available and the integer tableau is preserved. However, on the basis of the dual simplex transformations [Equations (3.6) – (3.8)], we note that a small λ will cause a larger improvement in the objective function that a larger λ since the change is a direct function of $b_{t0} = \left\lfloor \frac{x_{t0}}{\lambda} \right\rfloor < 0$. That is, the new value of the objective function is given by

$$x_{00} + x_{0k}b_{t0} = x_{00} + (z_k - c_k)\left\lfloor \frac{x_{t0}}{\lambda} \right\rfloor$$

where $\mathbf{P}_k$, the pivot column, must be the one such that

$$\theta = \min_{x_{tj} < 0} \frac{x_{0j}}{-1} = \frac{z_k - c_k}{-1}$$

(A $z_k - c_k = 0$ indicates degeneracy in the dual problem).

Since λ must be chosen such that $b_{lk} = \left\lfloor \frac{x_{lk}}{\lambda} \right\rfloor = -1$, we have from Equation (3.6)

$$\frac{z_k - c_k}{\left\lfloor \frac{x_{lk}}{\lambda} \right\rfloor} = \frac{z_k - c_k}{-1} \leq \frac{z_k - c_k}{\left\lfloor \frac{x_{lj}}{\lambda} \right\rfloor} = \frac{z_j - c_j}{b_{lj}}$$

or $\dfrac{z_k - c_k}{-1} \leq \dfrac{z_j - c_j}{-1}$

Let m_j be the largest integer for which

$$\frac{z_k - c_k}{-1} \leq \frac{z_j - c_j}{-m_j}$$

Then

$$-b_{lj} = -\left\lfloor \frac{x_{lj}}{\lambda} \right\rfloor \leq m_j \text{ for } x_{lj} < 0 \qquad (5.6)$$

Hence, for each j, the smallest λ that satisfies (5.6) and allows $\mathbf{P}_k$ to be the pivot column, i.e., yields a pivot element of $b_{lk} = -1$, is given by $\lambda_j = -\left(\frac{x_{lj}}{m_j}\right)$. As the minimum permissible λ must be at least as great as the largest λ_j, we have $\lambda_{\min} = \max\limits_{x_{lj} < 0} \lambda_j$. λ is not restricted to be an integer, and for $\mathbf{P}_k$ we have $m_k = 1$, $\lambda_k = -x_{lk}$, $\lambda_{\min} \geq \lambda_k$; hence $b_{lk} = \left\lfloor \frac{x_{lk}}{\lambda_{\min}} \right\rfloor = -1$ [6].

Example 5.1: min $z = 2x_1 + x_2$

$$\begin{array}{llll} \text{st} & x_1 + x_2 + x_3 & & = 4 \\ & -x_1 - x_2 & + x_4 & = -4 \\ & -2x_1 + x_2 & \quad + x_5 & = -3 \\ & x_j \geq 0 \text{ and } x_1, x_2 \text{ integer} & & \end{array}$$

The set of equations $x'_1 = x_1$, $x'_2 = x_2$ and the conditions $c'_j = c_j$ have been added in Tableau 5-2, and row 6 has been added for the cutting constraint and column $\mathbf{P}_6$ for the new variable.

The second equation of the system is used to generate the cutting constraint. We have

$$\frac{z_k - c_k}{-1} = \min_{x_{2j}<0} \frac{z_j - c_j}{-1} = \frac{z_2 - c_2}{-1} = 1$$

Hence column $\mathbf{P}_2$ is to be the pivot column. To determine the m_j, we have

$$1 \leq \frac{2}{m_1} \qquad 1 \leq \frac{1}{m_2}$$

Tableau 5-2. Initial tableau for Example 5.1.

				2	1	0	0	0	2	1	
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P'}_1$	$\mathbf{P'}_2$	$\mathbf{P}_6$
0	---	-	0	-2	-1	0	0	0	0	0	0
1	$\mathbf{P}_3$	0	4	1	1	1	0	0	0	0	0
2	$\mathbf{P}_4$	0	-4	-1	-1	0	1	0	0	0	0
3	$\mathbf{P}_5$	0	-3	-2	1	0	0	1	0	0	0
4	$\mathbf{P'}_1$	2	0	-1	0	0	0	0	1	0	0
5	$\mathbf{P'}_2$	1	0	0	-1	0	0	0	0	1	0
6	$\mathbf{P}_6$	0	-2	-1	**-1**	0	0	0	0	0	1

or $m_1 = 2$, $m_2 = 1$. The $\lambda_j = \frac{-x_{lj}}{m_j}$ are then

$$\lambda_1 = \frac{1}{2} \quad \lambda_2 = \frac{1}{1} = 1$$

and since $\lambda = \max \lambda_j = 1$, we choose $\lambda = 2$.

The new constraint is then given by

$$\left\lfloor \frac{-4}{2} \right\rfloor = \left\lfloor \frac{-1}{2} \right\rfloor x_1 + \left\lfloor \frac{-1}{2} \right\rfloor x_2 + x_6$$

$$-2 = -x_1 - x_2 + x_6$$

This equation is shown added to Tableau 5-2. By applying the elimination transformation with the pivot element as shown, Tableau 5-3 is obtained. Since all elements of column $\mathbf{P}_2$ have been reduced to zeros, it can be eliminated from the tableau and column $\mathbf{P}_6$ substituted in its place. In fact, since vectors $\mathbf{P}_3$, $\mathbf{P}_4$, $\mathbf{P}_5$, $\mathbf{P}'_1$, $\mathbf{P}'_2$ will not change under the all-integer algorithm transformation, the tableau can be made concise as in Tableau 5-4. Note that $x'_2 = x_2$ is still in the basis and now has a value

Tableau 5-3. Second step of Example 5.1.

				2	1	0	0	0	2	1	
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}'_1$	$\mathbf{P}'_2$	$\mathbf{P}_6$
0	---	-	2	-1	0	0	0	0	0	0	-1
1	$\mathbf{P}_3$	0	2	0	0	1	0	0	0	0	1
2	$\mathbf{P}_4$	0	-2	0	0	0	1	0	0	0	-1
3	$\mathbf{P}_5$	0	-5	-3	0	0	0	1	0	0	1
4	$\mathbf{P}'_1$	2	0	-1	0	0	0	0	1	0	0
5	$\mathbf{P}'_2$	1	2	1	0	0	0	0	0	1	-1
6											

Tableau 5-4. Simplified tableau form.

i	Basis	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_6$	$\mathbf{P}_7$
0	---	2	-1	-1	
1	$\mathbf{P}_3$	2	0	1	
2	$\mathbf{P}_4$	-2	0	-1	
3	$\mathbf{P}_5$	-5	**-3**	1	
4	$\mathbf{P}'_1$	0	-1	0	
5	$\mathbf{P}'_2$	2	1	-1	
6	$\mathbf{P}_7$	-2	-1	0	1

Tableau 5-5. Third step.

i	Basis	$\mathbf{P}_0$	$\mathbf{P}_7$	$\mathbf{P}_6$	$\mathbf{P}_8$
0	---	4	-1	-1	
1	$\mathbf{P}_3$	2	0	1	
2	$\mathbf{P}_4$	-2	0	**-1**	
3	$\mathbf{P}_5$	1	-3	1	
4	$\mathbf{P}'_1$	2	-1	0	
5	$\mathbf{P}'_2$	0	1	-1	
6	$\mathbf{P}_8$	-2	0	-1	1

of 2. We still have more than one row with a negative constant term, and we choose row 3. The pivot column is $\mathbf{P}_1$ and $\lambda = 3$. The new constraint is $-2 = -x_1 + x_7$. The new solution is shown in Tableau 5-5.

Selecting row 2 as the row to generate the cutting constraint, we have $\mathbf{P}_6$ as the pivot column and $\lambda = 1$, so again we choose $\lambda = 2$. The new constraint is $-2 = -x_6 + x_8$. The new solution is given by Tableau 5-6. With row 3 as the generating row, we have $\mathbf{P}_7$ as the pivot column and $\lambda = 3$. The new constraint is $-1 = -x_7 + x_9$. This yields Tableau 5-7. There are no negative constant terms, therefore we have reached the optimal all-integer solution.

Tableau 5-6. Fourth step.

i	Basis	$\mathbf{P}_0$	$\mathbf{P}_7$	$\mathbf{P}_8$	$\mathbf{P}_9$
0	---	6	-1	-1	
1	$\mathbf{P}_3$	0	0	1	
2	$\mathbf{P}_4$	0	0	-1	
3	$\mathbf{P}_5$	-1	**-3**	1	
4	$\mathbf{P}'_1$	2	-1	0	
5	$\mathbf{P}'_2$	2	1	-1	
6	$\mathbf{P}_8$	-1	-1	0	1

Tableau 5-7. Optimal solution.

i	Basis	$\mathbf{P}_0$	$\mathbf{P}_9$	$\mathbf{P}_8$
0	---	7	-1	-1
1	$\mathbf{P}_3$	0	0	1
2	$\mathbf{P}_4$	0	0	-1
3	$\mathbf{P}_5$	2	-3	1
4	$\mathbf{P}'_1$	3	-1	0
5	$\mathbf{P}'_2$	1	1	-1

The above procedure can be summarized by the following steps:

1. Assume that a dual feasible solution with an all-integer tableau can be determined.

2. From the rows containing a negative constant term, select a row to be used to generate the cutting constraint (the l^{th} row). If no such rows exist, an optimal all-integer solution has been found.

3. The pivot column corresponds to the $\mathbf{P}_k$ whose

   $$-(z_k - c_k) = \min_{x_{lj}<0} -(z_j - c_j) \quad \text{for } j \geq 1.$$

   If no such $x_{lj} < 0$ exists, then the problem is not feasible.

4. For each j having a $x_{lj} < 0$, determine the largest integer

$$-(z_k - c_k) \le \frac{-(z_j - c_j)}{m_j}$$

5. Define $\lambda_j = -\left(\frac{x_{lj}}{m_j}\right)$ for $x_{lj} < 0$.

6. Determine $\lambda_{\min}$ = max λ_j. If max $\lambda_j = 1$, set $\lambda_{\min} > 1$.

7. Develop the corresponding cutting constraint (5.5) and add it to Tableau 5-1, with the new equation being the $(n + 1)^{st}$ row and the new basic variable x_{n+s} corresponding to column $\mathbf{P}_{n+s}$ (where s is the index of the iteration).

8. Apply the simplex transformations to the new tableau using $b_{lk} = -1$ as the pivot element. This causes x_{n+s} to be a nonbasic variable, and because the pivot element was –1, the tableau remains in integers. The new row is then dropped from further consideration and the process repeated [6].

5.3 Gomory Mixed-Integer Method

In the previous method we assumed that all elements of the simplex tableau are integers. Gomory developed another other cutting plane methods in which the elements of the simplex tableau are not required to be integers. The method to be discussed is applicable to either pure or mixed integer programs.

Again we consider the general IP problem

$$\begin{aligned} \min\ z &= \mathbf{cx} \\ \text{st}\quad \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq 0 \\ x_j &\text{ integer}, j \in \mathrm{I} \end{aligned}$$

where the set I consists of the subscripts corresponding to those variables required to be integer.

Suppose that we have solved the LP relaxation, the l^{th} basic variable is not an integer, but is required to be an integer. Then we can write

$$x_l = a_{l0} - \sum_{j \in \mathrm{R}} a_{lj} x_j \tag{5.7}$$

where R = $\{j | x_j \text{ is nonbasic}\}$. In the current solution, all x_j, $j \in$ R, are zero, and so a_{l0} is not an integer. We will express a_{l0} as the sum of its integer part and its fractional part. Let $\lfloor a_{l0} \rfloor$ be the greatest integer less than or equal to a_{l0} and let f_0 be the remaining fractional part. Thus

$$a_{l0} = \lfloor a_{l0} \rfloor + f_0 \tag{5.8}$$

$$\lfloor a_{l0} \rfloor \geq 0 \tag{5.9}$$

$$0 < f_0 < 1 \tag{5.10}$$

Substituting (5.8) into (5.7) and rearranging gives

$$f_0 - \sum_{j \in \mathrm{R}} a_{lj} x_j = x_l - \lfloor a_{l0} \rfloor = \text{integer} \tag{5.11}$$

R is now divided into the nonnegative entries of the simplex tableau (R^+) and the negative entries of the simplex tableau (R^-) as follows:

$$R^+ = \{j|\ a_{ij} \geq 0, j \in R\}$$

$$R^- = \{j|\ a_{ij} < 0, j \in R\}$$

Then (5.11) becomes

$$f_0 - \sum_{j \in R^+} a_{ij}x_j - \sum_{j \in R^-} a_{ij}x_j = \text{integer} \tag{5.12}$$

Consider the two cases:

1. $\sum_{j \in R} a_{ij}x_j \geq 0$

2. $\sum_{j \in R} a_{ij}x_j < 0$

Case 1: $\sum_{j \in R} a_{ij}x_j \geq 0$

By (5.12), $\sum_{j \in N} a_{ij}x_j = f_0 + P$, where P = 0, 1, ... (i.e., some nonnegative integer). Hence any feasible solution to the IP problem must satisfy

$$\sum_{j \in R} a_{ij}x_j \geq f_0$$

Moreover, since $\sum_{j \in R^-} a_{ij}x_j \leq 0$, for any feasible solution, then any feasible solution to the IP problem also must satisfy

$$\sum_{j \in R^+} a_{ij}x_j \geq f_0 \tag{5.13}$$

Case 2: $\sum_{j\in R} a_{ij}x_j < 0$

From (5.11) it must be true that $\sum_{j\in R} a_{ij}x_j = f_0 - N$, where N = 1, 2, ... (i.e., some strictly positive integer). Then any feasible solution to the original problem must satisfy

$$\sum_{j\in R} a_{ij}x_j \leq f_0 - 1$$

since $f_0 - 1$ is the largest possible value for $\sum_{j\in R} a_{ij}x_j$. This can be rewritten as

$$\sum_{j\in R^+} a_{ij}x_j + \sum_{j\in R^-} a_{ij}x_j \leq f_0 - 1 \qquad (5.14)$$

Now multiply (5.14) by the strictly negative number $\frac{f_0}{f_0 - 1}$, and obtain

$$\sum_{j\in R^-} \left(\frac{f_0}{1 - f_0}\right) |a_{ij}| x_j \geq f_0 \qquad (5.15)$$

Since $\frac{f_0}{f_0 - 1}$ is a strictly negative number and $a_{ij} < 0$, for all *j*, then applying the negative sign from each a_{ij} to $\frac{f_0}{f_0 - 1}$ makes it $\frac{f_0}{1 - f_0}$. This makes each $a_{ij} > 0$ and is the reason for the notation $|a_{ij}|$ in (5.15).

Finally, combine the two inequalities (5.13) and (5.15) obtained for Cases 1 and 2, respectively, as follows:

$$\sum_{j\in R^+} a_{ij}x_j + \sum_{j\in R^-}\left(\frac{f_0}{1-f_0}\right)|a_{ij}|x_j \geq f_0 \tag{5.16}$$

Inequality (5.16) must be satisfied by any feasible solution to the original problem, regardless of whether Case 1 or Case 2 holds. This is true since the left-hand side is always positive and one or both of the two terms on the left-hand side are greater than or equal to f_0. Also, inequality (5.16) is not satisfied by the current optimal solution to the LP relaxation since all x_j, $j \in R^+$ or $j \in R^-$ are currently zero, and $f_0 > 0$ by definition. Thus (5.16) can be used as the cutting plane constraint.

However, in deriving (5.16) we did not take into account that some of the nonbasic variables may also be required to be integers. If we include this additional information the cutting plane may be even better than (5.16), i.e., the constraint will further restrict the feasible values of the nonbasic variables. The smaller the coefficient of a variable x_j in a particular constraint, the more the range for that variable is restricted, and the better the constraint is, with respect to that variable.

We now develop a cutting plane constraint in which the coefficient of each variable is as small as possible. First define

$R_1 = R \cap I = \{j|\ x_j$ is nonbasic and required to be an integer$\}$

$R_2 = R\backslash R_1 = \{j|\ x_j$ is nonbasic and not required to be an integer$\}$

$R_2^+ = R^+ \cap R_2 = \{j|\ a_{ij} \geq 0 \text{ and } j \in R_2\}$

$R_2^- = R^- \cap R_2 = \{j|\ a_{ij} < 0 \text{ and } j \in R_2\}$

As we did for a_{i0}, we let $a_{ij} = \lfloor a_{ij} \rfloor + f_j$, where $\lfloor a_{ij} \rfloor$ is an integer and $0 \le f_j \le 1$. Then, Equation (5.11) becomes

$$f_0 - \sum_{j \in R_1} f_j x_j - \sum_{j \in R_1} \lfloor a_{ij} \rfloor x_j - \sum_{j \in R_2} a_{ij} x_j = \text{integer}$$

Since $\sum_{j \in R_1} \lfloor a_{ij} \rfloor x_j$ must be an integer in any feasible solution to the original problem,

$$f_0 - \sum_{j \in R_1} f_j x_j - \sum_{j \in R_2} a_{ij} x_j = \text{integer} \tag{5.17}$$

Applying the same arguments to (5.17) which were used to obtain (5.16), we will obtain the following constraint:

$$\sum_{j \in R_1} f_j x_j + \sum_{j \in R_2^-} a_{ij} x_j + \sum_{j \in R_2^+} \left(\frac{f_0}{1 - f_0} \right) |a_{ij}| x_j \ge f_0 \tag{5.18}$$

We can obtain another cutting constraint by letting $\lceil a_{ij} \rceil$ be the smallest integer greater then a_{ij}, so that $a_{ij} - f_j + 1 = \lceil a_{ij} \rceil$. Then Equation (5.11) becomes

$$f_0 - \sum_{j \in R_1} (\lceil a_{ij} \rceil + f_j - 1) x_j - \sum_{j \in R_2} a_{ij} x_j = \text{integer}$$

or

$$f_0 - \sum_{j \in R_1} (f_j - 1) x_j - \sum_{j \in R_2} a_{ij} x_j = \text{integer} \tag{5.19}$$

since $\sum_{j \in R_1} \lceil a_{ij} \rceil x_j$ must be an integer in any feasible solution to the original problem. Note that $\sum_{j \in R_1} (f_j - 1)x_j \leq 0$. Apply the same arguments to (5.18) that were applied to (5.17) to obtain the constraint

$$\sum_{j \in R_1} \left(\frac{f_0}{1 - f_0}\right)(1 - f_j)x_j + \sum_{j \in R_2} a_{ij}x_j + \sum_{j \in R_2^+} \left(\frac{f_0}{1 - f_0}\right)|a_{ij}|x_j \geq f_0 \qquad (5.20)$$

Comparing the coefficients of (5.18) and (5.20) we see that, for $j \in R_1$, we can choose either f_j or $\frac{f_0(1 - f_j)}{(1 - f_0)}$ as the coefficient of x_j, $j \in R_1$.

Since the goal is to make the coefficient of each x_j as small as possible, we can choose the smallest of these two quantities.

Note that if $\frac{f_0(1 - f_j)}{(1 - f_0)} > f_j$ then $f_0 - f_0f_j < f_j - f_jf_0 \Rightarrow f_0 < f_j$

Hence f_j is the smaller of the two if $f_j > f_0$. Therefore, we can obtain an even better cutting plane constraint by combining the above, and choosing the following constraint [3]:

$$\sum_{j \in R} d_{ij}x_j \geq f_0 \qquad (5.21)$$

where

$$d_{ij} = \begin{cases} a_{ij}, & j \in R_2^+ \\ \left(\dfrac{f_0}{1-f_0}\right)|a_{ij}|, & j \in R_2^- \\ f_j, & j \in R_1 \text{ if } f_j \le f_0 \\ \dfrac{f_0(1-f_j)}{(1-f_0)}, & j \in R_1 \text{ if } f_j > f_0 \end{cases} \tag{5.22}$$

Suppose all the nonbasic variables in a current optimal simplex tableau are integer constrained. Then the sets R_2^+ and R_2^- would be empty, and the mixed-integer Gomory cut for that tableau reduces to

$$x^M = -f_0 + \sum_{j \in R_1^-} f_j x_j + \sum_{j \in R_1^+} \frac{f_0(1-f_j)}{(1-f_0)} x_j \ge 0, \tag{5.23}$$

where $R_1^- = \{j | j \in R_1 \text{ and } f_j \le f_0\}$ and $R_1^+ = \{j | j \in R_1 \text{ and } f_j > f_0\}$. Since the x_j, $j = 1, 2, \ldots$, are integer constrained we could use the classical integer cut

$$x^I = -f_0 + \sum_{j=1}^{n} f_j x_j \ge 0 \tag{5.24}$$

in place of (5.23). However, inequality (5.23) is usually stronger than (5.24).

Theorem 5.1. Suppose that the x_j, $j = 1, 2, \ldots, n$ are integer variables and R_1^+ in (5.23) is not empty. Then $x^M \ge 0$ is a stronger inequality than $x^I \ge 0$.

Proof: Since the union of the sets R_1^- and R_1^+ contains all the indices $j = 1, 2, ..., n$, we have

$$x^I = -f_0 + \sum_{j\in R_1^-} f_j x_j + \sum_{j\in R_1^+} f_j x_j \geq 0$$

Also

$$x^M = -f_0 + \sum_{j\in R_1^-} f_j x_j + \sum_{j\in R_1^+} \frac{f_0(1 - f_j)}{(1-f_0)} x_j \geq 0$$

Now, if $j \in R_1^+$, $f_j > f_0$; or

$$f_0\left(\frac{1-f_j}{1-f_0}\right) < f_0 < f_j \text{ since } \frac{1-f_j}{1-f_0} < 1.$$

Thus, x^M is a stronger inequality than x^I because it has smaller coefficients of x_j when $j \in R_1^+$. That is, for $j \in R_1^+$, the hyperplane $x^M = 0$ intersects the coordinate axis x_j at a distance farther from the origin than the corresponding intersection with the hyperplane $x^I = 0$. For $j \in R_1^-$, the coefficients are the same and hence the hyperplanes intersect each of the x_j axis at the same point (Salkin[15]). ♦

Example 5.2: min $z = -5x_1 + 6x_2 + 9x_3$

$$\begin{aligned} \text{st}\quad & 9x_1 + 10x_2 + 6x_3 + x_4 = 50 \\ & 6x_1 + 3x_2 + 19x_3 + x_5 = 35 \\ & x_1, x_2, x_3 \geq 0 \\ & x_1, x_2 \text{ integer} \end{aligned}$$

To begin, we set up initial Tableau 5-8. We then solve the LP relaxation and obtain the optimal LP solution, as shown in Tableau 5-9. Since x_1 must be an integer, we choose row 1 to generate the cut and

$$a_{10} = \frac{50}{9} = 5 + \frac{5}{9} \Rightarrow f_0 = \frac{5}{9}$$

thus

$$\frac{f_0}{1-f_0} = \frac{\frac{5}{9}}{1-\frac{5}{9}} = \frac{5}{4}$$

Using (5.22) we obtain the cutting plane constraint

Tableau 5-8. Initial tableau for Example 5.2.

				-5	6	9	0	0
i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$
0	---	-	0	5	-6	-9	0	0
1	$\mathbf{P}_4$	0	50	9	10	6	1	0
2	$\mathbf{P}_5$	0	35	6	3	19	0	1

Tableau 5-9. Optimal LP solution.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$
0	---	-	$-250/9$	0	$-104/9$	$-37/3$	$-5/9$	0
1	$\mathbf{P}_1$	-5	$50/9$	1	$10/9$	$2/3$	$1/9$	0
2	$\mathbf{P}_5$	0	$5/3$	0	$-11/3$	15	$-2/3$	1

$$\frac{1}{9}x_2 + \frac{2}{3}x_3 + \frac{1}{9}x_4 \geq \frac{5}{9}$$

or

$$x_2 + 6x_3 + x_4 \geq 5$$

x_4 is a slack variable, so we substitute and get

$$x_2 + 6x_3 + (50 - 9x_1 - 10x_2 - 6x_3) \geq 5$$

$$\Rightarrow x_1 + x_2 \leq 5$$

$$\Rightarrow x_1 + x_2 + x_6 = 5$$

We add this to the optimal LP tableau (and subtract row 1 from row 3 to have a full basis) to obtain Tableau 5-10. Performing one dual simplex iteration gives the optimal mixed-integer solution shown in Tableau 5-11.

Tableau 5-10. Constraint added to tableau.

i	Basis	c	P_0	P_1	P_2	P_3	P_4	P_5	P_6
0	---	-	$-250/9$	0	$-104/9$	$-37/3$	$-5/9$	0	0
1	P_1	-5	$50/9$	1	$10/9$	$2/3$	$1/9$	0	0
2	P_5	0	$5/3$	0	$-11/3$	15	$-2/3$	1	0
3	P_6	0	$-5/9$	0	$-1/9$	$-2/3$	$-1/9$	0	1

Tableau 5-11. Optimal mixed-integer solution.

i	Basis	c	P_0	P_1	P_2	P_3	P_4	P_5	P_6
0	---	-	-25	0	-11	-9	0	0	-5
1	P_1	-5	5	1	1	0	0	0	1
2	P_5	0	5	0	-1	19	0	1	-6
3	P_4	0	5	0	1	6	1	0	-9

CHAPTER 6

BRANCH-AND-BOUND

6.1 Introduction

The idea of using a branch-and-bound algorithm for integer programming using LP relaxations was proposed by Land and Doig during the 1960's and was later modified by Dakin. However, the term branch-and-bound was coined by Little *et al* in their study of such an algorithm to solve the traveling salesman problem (Linderoth and Savelsbergh [10]).

Branch-and-bound has two appealing qualities. First, it can be applied to the mixed and pure IPs in essentially the same way, so a single algorithm works for both. Second, the branch-and-bound typically yields a succession of feasible integer solutions (as opposed to traditional cutting plane approaches in which feasibility is not obtained until the problem is solved), so that if the computations are terminated due to time or memory restrictions, the current best solution becomes a candidate for the optimum [1].

The effectiveness of the branch-and-bound procedure for solving mixed IPs using LP relaxation is well documented. After the introduction of this procedure in the 1960's, researchers in the 1970's examined strategies for searching the branch-and-bound tree in an efficient manner. With a few exceptions, research in the past decades has strayed from developing and examining effective search strategies and instead focused on improving the bound obtained by the LP relaxation [10]. In fact, the main use of the LP relaxation in solving a mixed IP is that the optimal value of the LP relaxation provides a lower bound (for the minimization problem) on the optimal value of the corresponding mixed IP (Johnson et al [9]).

For pure and mixed IPs many of the most effective branch-and-bound procedures are based on the use of the simplex method. Simplex based methods begin by solving the LP relaxation of the problem. If an integer solution is not obtained, one of the integer variables that is fractional in the LP solution, denoted by x_k, where k is the variable index, is selected and two descendants of the original problem are created. This operation is called *branching*. Thus the original problem has been replaced by two IPs that now must be solved [1].

More formally, let z^* denote the objective function value of the best integer solution found thus far (the incumbent) and z_{LP} the objective function value of the corresponding LP relaxation. Then, whenever a current IP minimization problem is solved as an LP, one of the following occurs:

1.) The LP has no feasible solution (in which case the current IP also has no feasible solution).

2.) The LP has an optimal solution $z_{LP} \geq z^*$ (in which case the current IP optimum $z_{IP} \geq z_{LP} \geq z^*$ and therefore cannot provide an improvement over the incumbent).

3.) The optimal solution to the LP is integer and feasible, and yields $z_{LP} < z^*$ (in which case the solution is optimal for the current IP and provides an improved incumbent for the original IP. Therefore, z^* is reset to z_{LP}).

4.) None of the above occurs; i.e., the optimal LP solution exists, satisfies $z_{LP} < z^*$, but is not integer and hence not feasible to the current IP.

In each of the first three cases, the current IP (the current node) is said to be *fathomed* or *implicitly enumerated.* A node can be fathomed by infeasibility (case 1), bound (case 2), or optimality (case 3). A problem fathomed as a result of (3) yields useful information because it allows the incumbent to be updated. If the problem is not fathomed and hence winds up in (4), further exploration or branching is required. A node that is not fathomed and whose corresponding constraint set has not been divided is said to be *live* or *active.* If there are no active nodes remaining the enumeration is complete [1].

A tree of solutions, as shown in Figure 6-1, may describe the set of all solutions to a problem solved in this way. This figure shows the case of a 0-1 problem.

In the figure, the numbered circles are called *nodes* or *subproblems*, and the lines connecting the nodes are called *branches* or *edges*. Node 0, the root node, represents the set of all solutions. A sequence of nodes and branches from node 0 to any other node *k* is called a *path* to node *k* and will be denoted by P_k. Along each branch one constraint is imposed on the variables, so that node *k* represents the set of all solutions which satisfy the original constraints plus the constraints imposed by each of the branches in the path from node 0 to node *k*. For example, node 4 of Figure 6-1 represents the set of all solutions for which $x_1 = 0$, $x_2 = 1$, in addition to the original constraints. A node further down a given path represents a subset of solutions of any node above it on the same path [3].

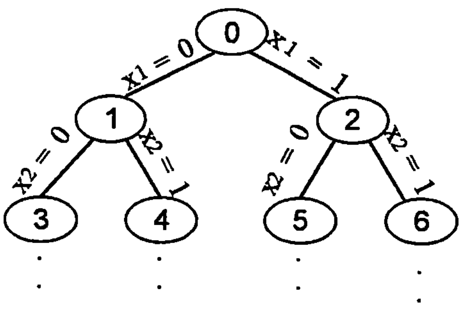


Figure 6-1. Tree of solutions for 0-1 programming problem.

6.2 Method Using LP Relaxation

We will denote the feasible region for the IP by S = {$\mathbf{x}$: $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq 0$ and integer}. The feasible region for the corresponding LP relaxation will be denoted by T = {$\mathbf{x}$: $\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq 0$}, where $S \subseteq T$. For a path P_k in the tree connecting node 0 to node k, let S^k be the intersection of S with the set of points satisfying the constraints given by the edges of P_k. Note that $S^k \subseteq S$.

Suppose that the enumeration is at node i in the tree. The problem at node i is

$$z^i_{IP} = \min \{\mathbf{cx}: \mathbf{x} \in S^i\} \qquad (6.1)$$

A lower bound $\underline{z}^i \leq z^i_{IP}$ may be calculated by considering the relaxation (LP^i) of (6.1):

$$\underline{z}^i = z^i_{LP} = \min \{\mathbf{cx}: \mathbf{x} \in T^i \supseteq S^i\} \qquad (6.2)$$

Note that a lower bound at node is valid for any of its successors; i.e., if node k is a successor of node i, then $T^i \supseteq S^i \supseteq S^k$.

Suppose that the LP (6.2) is solved at node i and the solution $\mathbf{x}^i$ is not all-integer. In particular, some basic variable $x_k = \lfloor x_k \rfloor + f_k$, $0 < f_k < 1$. Then a partition of S^i is

$$\left\{S^i \cap \{x: x_k \leq \lfloor x_k \rfloor\}, S^i \cap \{x: x_k \geq \lceil x_k \rceil\}\right\} \qquad (6.3)$$

where $\lfloor x_k \rfloor$ denotes the largest integer less than or equal to x_k, and $\lceil x_k \rceil$ denotes the smallest integer greater than or equal to x_k.

Assume that an upper bound u_j is known for each variable x_j, $j = 1, \ldots, n$. Let

$$S^i = \left\{\mathbf{x}: \mathbf{Ax} = \mathbf{b}, 0 \le \alpha_j^i \le x_j \le \beta_j^i \le u_j, x_j \text{ integer}, j = 1, \ldots, n\right\} \quad (6.4)$$

at node i, where α_j^i and β_j^i are integers determined from (6.3), and $\alpha_j^0 = 0$ and $\beta_j^0 = u_j$.

The LPs resulting from the partition (6.3) are manageable since the added constraints are lower and upper bounds on the individual variables. These problems are solved using either relaxation or the dual simplex algorithm (Wolsey [19]). We will use relaxation.

The algorithm is as follows:

Step 1 (Initialization): Begin at node i = 0, where S^0 is given in (6.4), $\underline{z}^0 = -\infty$ and $\overline{z^0} = \infty$. Go to Step 2.

Step 2 (Branching): If no active nodes exist, go to Step 7; otherwise, select an active node i. If the LP (6.2) has been solved, go to Step 3; otherwise, go to Step 4.

Step 3 (Partitioning): Choose a variable x_k with $f_k > 0$ and partition S^i as in (6.3). Go to Step 2.

Step 4 (Solving LP): Solve the LP relaxation (6.2). If the LP is not feasible, fathom node i and go to Step 2. If the LP has an optimal solution **x**^{i}, let $\underline{z}^i = \lceil z^i_{LP} \rceil$ and go to Step 5. (Note that $\underline{z}^i$ can be rounded up because every integer solution yields an integer objective function value, assuming that the cost coefficients are integers, as is done here.)

Step 5 (Fathoming by optimality): If x^i is not all-integer, go to Step 6; otherwise, let $\overline{z^i} = z^i$ and fathom node i. Let $\overline{z^0} = \min\{\overline{z^0}, \overline{z^i}\}$ and go to Step 6.

Step 6 (Fathoming by bounds): Fathom any node i such that $\underline{z}^i \geq \overline{z^0}$ and go to Step 2.

Step 7 (Termination): Terminate. If $\overline{z^0} = \infty$, there is no feasible solution. If $\overline{z^0} < \infty$, the feasible solution that yielded $\overline{z^0}$ is optimal. [1]

The algorithm generates a tree of solutions similar to that in Figure 6-1. Each non-terminal node has two branches, corresponding to the two constraints $x_k \leq \lfloor x_k \rfloor$ and $x_k \geq \lceil x_k \rceil$. However, unlike Figure 6-1, the same variable x_k may be constrained by more than one pair of branches.

Example 6.1: The use of the algorithm is illustrated in the example below:

$$\begin{aligned} \min z &= x_1 + 8x_2 \\ \text{st} \quad 2x_1 + 3x_2 &\geq 6 \\ -3x_1 + x_2 &\leq 1 \\ x_1 + x_2 &\leq 7 \\ x_1 - 3x_2 &\leq 1 \\ x_j \geq 0 \text{ and } x_1, x_2 & \text{ integer} \end{aligned}$$

To begin, the only active node is node 0 so we go to Step 4 and solve the corresponding LP. The optimal LP solution is $z^0_{LP} = {}^{53}\!/_{9}$, so we can set $\underline{z_0} = 6$. The decision vector $\mathbf{x}^0 = ({}^{7}\!/_{3}, {}^{4}\!/_{9})$ which is not all-integer. Following the logic, we go to Step 3 and choose x_1 for partitioning. Two new nodes are created as a result. The first, node 1, is associated with $x_1 \leq 2$ and the second, node 2, with $x_1 \geq 3$.

Solving the augmented LP at node 2 gives $z^2_{LP} = {}^{25}\!/_{3}$ and $\mathbf{x}^2 = (3, {}^{2}\!/_{3})$ which does not provide an improvement in the objective function lower bound because $\underline{z^2} = \lceil z^2_{LP} \rceil = 9$. We now choose x_2 for partitioning and create node 3 and node 4 defined by the constraints $x_2 \leq 0$ and $x_2 \geq 1$, respectively. Solving the LP at node 4 gives an all-integer solution $\mathbf{x}^4 = (3, 1)$ with $z^4_{LP} = 11$. We can set $\overline{z^0} = 11$ at Step 5 and fathom the node.

Nodes 1 and 3 remain active. We branch to node 3, the most recently created of the two. The LP is infeasible, so we fathom node 3.

Now only node 1 remains active. We go to Step 4. Solving the LP yields $z_{LP}^{1} = 22/3$ and decision vector $\mathbf{x}^1 = (2, 2/3)$. We branch on x_2, the only fractional variable, and create nodes 5 and 6.

Choosing node 5 results in an infeasible solution, and so, node 5 is fathomed. Following the logic takes us to Step 4. Solving the LP relaxation of node 6, we obtain $z_{LP}^{6} = 19/2$ and $\mathbf{x}^6 = (3/2, 1)$. Branching on x_1 creates new nodes 7 and 8 corresponding to $x_1 \leq 1$ and $x_1 \geq 2$, respectively.

We solve node 7 first. Solving the LP gives $z_{LP}^{7} = 35/3$ with $\mathbf{x}^7 = (1, 4/3)$ which is fractional. However, $\underline{z}^7 = \lceil z_{LP}^7 \rceil = 12$, so at Step 6 we see that $\underline{z}^7 > \overline{z^0}$ indicating that node 7 is fathomed. Branching on the ≥ constraint and solving the LP gives an all-integer solution $\mathbf{x}^8 = (2,1)$ with $z_{LP}^{8} = \overline{z^8} = 10$. At Step 5, the upper bound is updated to give $\overline{z^0} = \min\{11, 10\} = 10$ and node 8 is fathomed. No active nodes remain, so the optimal solution is $z^* = 10$ and $x^* = (2, 1)$.

The full search tree is shown in Figure 6-2. If we had branched off of variable x_2 instead of x_1 at node 0, we would have obtained the optimal solution much more quickly, as shown in Figure 6-3.

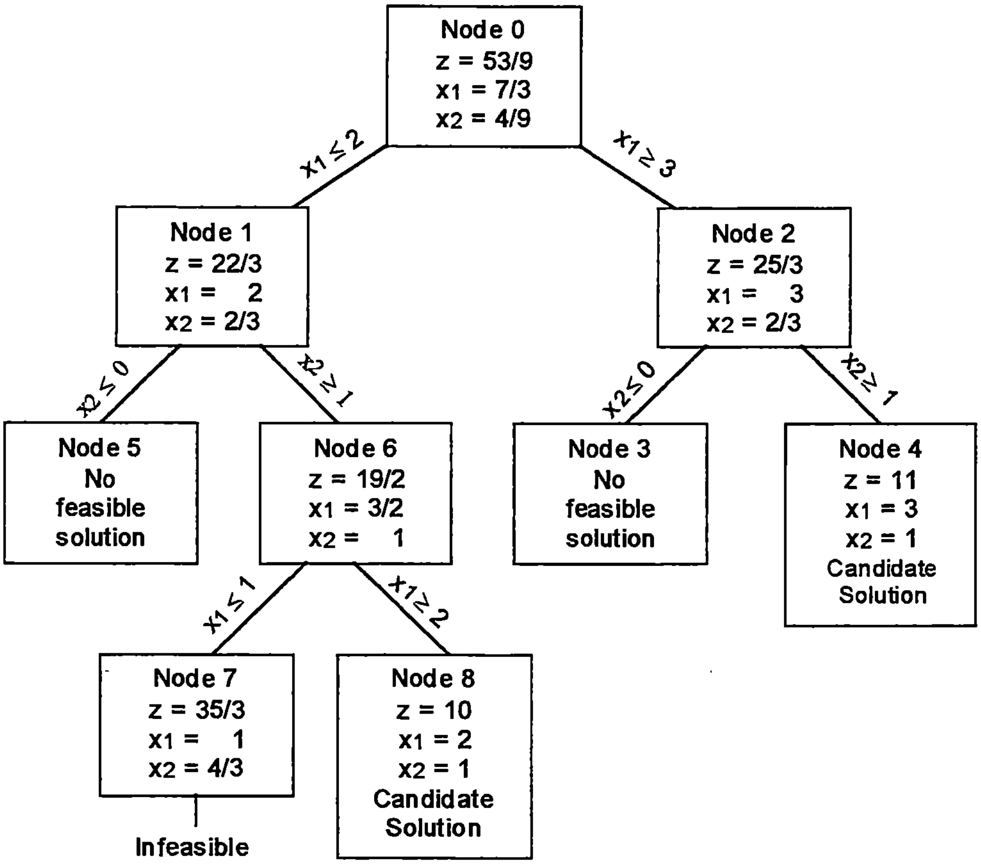


Figure 6-2. Tree of solutions for initial branch on x_1.

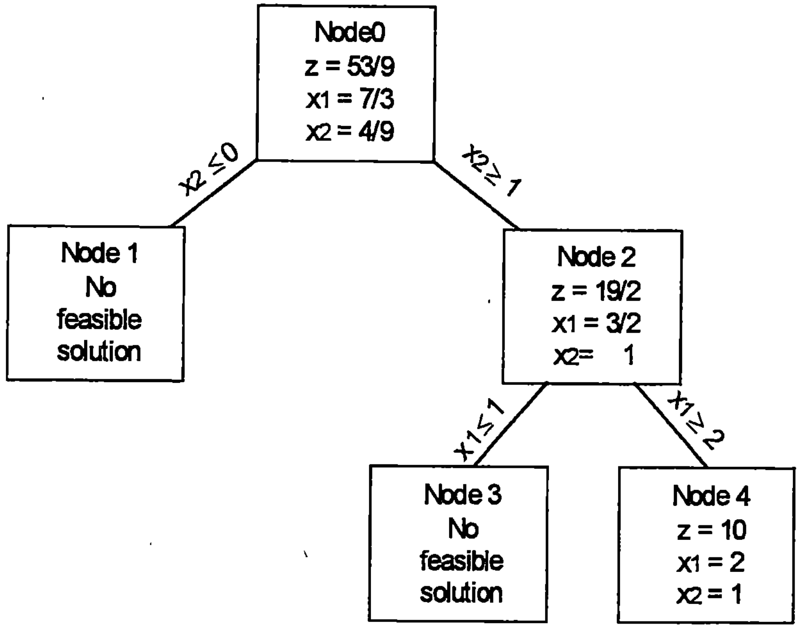


Figure 6-3. Tree of solutions for initial branch on x_2.

6.3 Branching

As can be seen from Figures 6-2 and 6-3, variable choice can be critical in keeping the tree size small. A simple rule is to branch on the variable whose fractional value is closest to $\frac{1}{2}$, i.e., with the maximum integer infeasibility. We can also assign priorities to the integer variables such that we branch on the most important variable first. The importance of an integer variable may be based on one or more of the following criteria:

- It represents an important decision in the model
- Its cost or profit coefficient in the objective function is very large compared to the others

❖ Its value is critical to the model based on the experience of the user (Ravindran[14])

Other rules, used by many commercial solvers, try to choose a variable that causes the LP objective function to increase quickly (for a minimization problem) in order to make the LP values at the child nodes as large as possible. This is done in an attempt to prune (fathom) one or both of the child nodes using the fact that a node can be pruned if its LP value is greater than or equal to the best known IP solution. Early branching on variables that cause big changes can be critical in keeping the tree small.

However, it would be too expensive to compute the actual objective function changes for all candidate variables every time that branching was required. Instead, estimates of the rate of objective function change on both the down and up branches, called *pseudocosts*, are used. The pseudocosts can be obtained from dual information at the node, or be based on actual changes from previous branchings involving the variable.

Instead of using an estimate of the change, we can also perform explicitly one or more dual simplex iterations to obtain a lower bound on the change, and use the lower bound to choose among variables. A recent variation of this idea has become known as *strong branching*. In strong branching, a fairly large number of dual simplex iterations are carried out, but only for a small number of candidate branching variables. Strong branching has been shown to be very effective for large set partitioning problems and large traveling salesman problems [9].

6.4 Node Selection

Another question that must be answered is, given a list L of active subproblems or equivalently, a partial tree of unfathomed or live nodes, which node should be examined next? Node choice rules are partially motivated by the desire to find good feasible solutions early in the search. This is important for two reasons: if time runs out a good answer may be all one gets, and a good upper bound inhibits the growth of the tree [9]. The available options can be divided into two categories: (1) a priori rules that determine in advance the order in which the tree will be developed; and (2) adaptive rules that select a node using information such as bounds or function values associated with the live nodes.

The most commonly used a priori type rule is *depth-first search plus backtracking,* which is also known as *last-in, first-out* (LIFO). In this approach, if the current node is not fathomed, the next node considered is one of its two children. *Backtracking* means that when a node is fathomed we go back on the path from this node toward the root until we come to the first unfathomed node which has a child node that is still active.

A second (basically) a priori rule is known as *breadth-first search* which is the opposite of depth-first search. The level of a node in an enumeration tree is the number of edges in the unique path between it and the root. In breadth-first search, all nodes at a given level are considered before any nodes at the next level. If tight lower and upper bounds are available early on, this approach can be very effective because it may significantly reduce the size of the overall tree. The number of new nodes potentially doubles with each successive level. This method of searching is also known as *best-first* or *best-bound* search [1].

Both methods have advantages and disadvantages. Best-first search tends to minimize the number of nodes evaluated and at any point during the search attempts to improve the global lower bound on the problem. Best-first search concentrates on proving that no solution better than the current one exists. Searching in this way may be restricted by memory requirements if good upper bounds are not found early, leading to relatively little pruning of the tree. In addition, since the tree is explored in a breadth-first fashion, with one LP having little relation to the next, computation times are higher.

Depth-first search overcomes the problems of breadth-first search. Searching in this way tends to minimize memory requirements, and changes in the LP from one node to the next are minimal – usually changing only one variable's bound. Another advantage is in finding feasible solutions since feasible solutions tend to be found deep in the search tree. Depth-first search was the method proposed by Dakin in 1965 and Little *et al* in 1963, primarily due to the small memory capabilities of computers at that time. Although this method of searching has its strengths, it can lead to lead search trees. The large trees are due to the fact that we may examine many nodes that would have been fathomed had a better upper bound $\overline{z^0}$ been known [10].

6.5 Preprocessing

Preprocessing applies simple logic to reformulate the problem and tighten the LP relaxation. In the process, preprocessing may also reduce the size of the problem by fixing variables and eliminating constraints. Sometimes preprocessing may detect infeasibility.

The simplest logical testing is on bounds. Let L_i be any lower bound on the value of the i^{th} row $A^i x$ subject only to $l \leq x \leq u$ and let U_i be any upper bound on the value of the i^{th} row $A^i x$ subject only to $l \leq x \leq u$. A constraint $A^i x \leq b_i$ is redundant if $U_i \leq b_i$ and is infeasible if $L_i > b_i$. A bound on a variable may be tightened by recognizing that a constraint becomes infeasible if the variable is set at that bound. In the case of 0-1 variables, the initial lower and upper bounds on the variable are 0 and 1 respectively. If these bounds are improved, then the variable can be fixed. For example, if the upper bound of a 0-1 variable is less than 1, then it can be fixed to 0. Of course, if the lower bound is positive and the upper bound is less than 1, then the problem is integer infeasible. In summary, considering how one row together with upper and lower bounds may lead to dropping the row if it is redundant, declaring the problem infeasible if that row is infeasible, or to tighten bounds on the variables.

These logical testing methods can become powerful when combined with *probing*. Probing means setting temporarily a 0-1 variable to 0 or 1 and then redoing the logical testing. If the logical testing shows that the problem has become infeasible, then the variable on which we probe can be fixed to its other bound. For example, $6x_1 - 4x_2 \leq 5$ becomes infeasible when x_2 is set to 0. We conclude that x_2 must be 1 in every feasible solution. If the logical testing results in another 0-1 variable being fixed, then a logical implication has been found. Consider $6x_1 + 5x_2 + 2x_3 \leq 9$. If x_1 is set to 1, then subsequent bound reduction will fix x_2 to 0. Thus, we have found the logical implication $x_1 = 1$ implies $x_2 = 0$, which can be represented by the inequality $x_1 + x_2 \leq 1$. Adding this inequality tightens the LP relaxation since $(1, \frac{3}{4}, 0)$ is feasible for the original inequality, but infeasible for the implication inequality. If the logical testing shows that a constraint has become redundant, then it can be tightened by what is called *coefficient reduction* or *coefficient improvement*. For example, $2x_1 + x_2 + x_3 \geq 1$ becomes redundant when x_1 is set to 1. Whenever a variable being set to 1 leads to a strictly redundant $\geq$ constraint, then the coefficient of the variable can be reduced by the amount that the constraint becomes redundant. Therefore, $2x_1 + x_2 + x_3 \geq 1$ can be tightened to $x_1 + x_2 + x_3 \geq 1$. Note that $(\frac{1}{2}, 0, 0)$ is no longer feasible to the LP relaxation of the tightened constraint. Less obvious coefficient improvements can also be found during probing. For example, if there are inequalities

$$x_1 \leq x_2,\ x_1 \leq x_3, \text{ and } x_2 + x_3 \geq 1 \tag{6.4}$$

then setting x_1 to 1 leads to a strictly redundant inequality $x_2 + x_3 > 1$, and the coefficient of x_1, namely 0, can be lowered to -1. Clearly, (6.4) and

$$x_1 \leq x_2,\ x_1 \leq x_3, \text{ and } -x_1 + x_2 + x_3 \geq 1$$

have the same set of 0-1 solutions. However, the fractional solution $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$ is allowed by the first set of inequalities, but not by the strengthened second set. One important aspect of probing and coefficient improvement is that it applies to the mixed 0-1 case and not just to pure 0-1 problems.

To give some idea of the relative computational difficulty of the various preprocessing techniques, any of the techniques applied to only one row is not likely to take much time and will almost always be worth doing. This remark applies to fixing, identifying redundant rows, bound reduction, coefficient improvement, and finding implication inequalities.

Full probing on the matrix is the most time consuming part of preprocessing. Probing may be restricted to probing on variables that are fractional in the current LP solution. Despite efficient implementations probing remains a possibly effective but sometimes too time consuming preprocessing technique that requires care in practice.

In summary, preprocessing may identify infeasibility, may identify redundant constraints, may improve bounds, and may fix variables. Coefficient improvement leads to simply replacing an inequality by a stronger one. Thus, it can be done in a preprocessing phase independent of solving the linear program [9].

CHAPTER 7

ADDITIVE ALGORITHM

7.1 Introduction

Before discussing the additive algorithm, we show how any pure IP may be expressed as a 0-1 IP: Simply express each variable in the original IP as the sum of powers of 2. For example, suppose that the variable x_i is required to be an integer. Let n be the smallest integer such that we can be sure that $x_i < 2^{n+1}$. Then x_i may be (uniquely) expressed as the sum of $2^0, 2^1, \ldots, 2^{n-1}, 2^n$, and

$$x_i = u_n 2^n + u_{n-1} 2^{n-1} + \cdots + u_2 2^2 + 2u_1 + u_0 \qquad (7.1)$$

where $u_i = 0$ or 1, $i = 0, 1, \ldots, n$.

To convert the original IP to a 0-1 IP, replace each occurrence of x_i by the right-hand side of (7.1). For example, suppose we know that $x_i \leq 50$. Then $x_i < 2^{5+1} = 64$, and (7.1) yields

$$x_i = 32u_5 + 16u_4 + 8u_3 + 4u_2 + 2u_1 + u_0 \qquad (7.2)$$

where $u_i = 0$ or 1, $i = 1, \ldots, 5$. Then replace each occurrence of x_i by the right-hand side of (7.2).

How can we find the values of the u's corresponding to a given value of x_i? Suppose $x_i = 43$. Then u_5 will be the largest multiple of $2^5 = 32$ that is contained in 43. This yields $u_5 = 1$; then the rest of the right side of (7.2) must equal 43 – 32 = 11. Then u_4 will be the largest multiple of $2^4 = 16$ contained in 11. This yields $u_4 = 0$. Then u_3 will be the largest multiple of $2^3 = 8$ contained in 11. This yields $u_3 = 1$. Continuing in this way, we obtain $u_2 = 0$, $u_1 = 1$, and $u_0 = 1$. Thus $43 = 2^5 + 2^3 + 2^1 + 2^0$.

Since 0-1 IPs are generally easier to solve than other pure IPs, one may ask why we don't transform every pure IP into a 0-1 IP. The answer is simply that the transformation from a pure IP into a 0-1 IP greatly increases the number of variables. However, problems such as lockbox and knapsack problems naturally yield 0-1 problems. Thus, it is worthwhile to learn how to solve 0-1 IPs [18].

The (mixed or pure) 0-1 linear problem has an important property characterized by the following theorem from Taha [17]:

Theorem 7.1. In a linear 0-1 problem, mixed or pure, assume that the integer constraint is replaced by the continuous range $0 \leq x_j \leq 1$; then

(i) the resulting continuous solution space contains no feasible points (i.e. no points satisfying the integer conditions) in its *interior*, and

(ii) the optimum feasible solution of the integer problem occurs at an extreme point of the (continuous) convex space.

Proof: Part (i) follows from the fact that a feasible point must satisfy the restriction $x_j = 0$ or $x_j = 1$. Since these are boundary planes of the continuous solution space, it is impossible, by definition, that any such point can be an interior point.

Part (ii) is obvious for the pure 0-1 problem. In the mixed case, suppose $x_j = x_j^*$, where $x_j^* = 0$ or 1, $j \in I$ (I is the set of subscripts of those variables required to be integer valued), are the optimal feasible integer values. By fixing these values, the resulting 0-1 problem becomes a regular LP in the remaining continuous variables. Thus, by LP theory, the optimum solution to the original 0-1 problem must occur at an extreme point of the (continuous) convex solution space, specifically because $x_j = x_j^*$ are *boundary* planes of the convex set. ♦

0-1 programming problems could be solved by using either the cutting plane method or the branch and bound method. The desire for greater efficiency and the simplicity of 0-1 problems led to the development of another method. The fact that each variable in a 0-1 problem can take on only one of two values means that small 0-1 problems can sometimes be solved by inspection. Consider the problem

$$\min z = 3x_1 + 5x_2 + 4x_3$$

$$\text{st} \quad x_1, x_2, x_3 = 0 \text{ or } 1$$

Clearly to minimize *z* we would make each variable as small as possible yielding the solution $x_1 = x_2 = x_3 = 0$.

Not all problems can be solved by inspection so easily. However, in a large problem it is possible that solution values for several of the variables can be found by simple inspection of the constraints. Once values are known for some variables, these variables are eliminated and the problem shrinks in size. Consider the problem

Example 7.1: $\min z = 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5$

$$\text{st} \quad x_1 - 3x_2 + 5x_3 + x_4 - 4x_5 \geq 2$$

$$2x_1 - 6x_2 + 3x_3 + 2x_4 - 2x_5 \leq 0$$

$$- x_2 + 2x_3 - x_4 - x_5 \geq 1$$

We want to identify those variables that cannot be 0 because of one or more constraints, and raise them to 1. Hopefully, one or another of the constraints enables us to identify one or more variables that must be 1 rather than 0. We construct from each constraint a bound on every variable that is a part of that constraint. Then we identify for each bound of the ≥ type a minimum lower bound by giving the variables on the right side of the inequality a value of 1 if the coefficient is negative and 0 if the coefficient is positive. For each bound of the ≤ type a maximum upper bound is identified by assigning a value of 1 to variables with positive coefficients and 0 otherwise.

Following this procedure we obtain the bounds shown below.

Constraint 1

(a) $x_1 \geq 2 + 3x_2 - 5x_3 - x_4 + 4x_5$; min LB = -4

(b) $x_2 \leq {}^{-2}\!/_3 + {}^{1}\!/_3 x_1 + {}^{5}\!/_3 x_3 + {}^{1}\!/_3 x_4 - {}^{4}\!/_3 x_5$; max UB = ${}^{5}\!/_3$

(c) $x_3 \geq \frac{2}{5} - \frac{1}{5}x_1 + \frac{3}{5}x_2 - \frac{1}{5}x_4 + \frac{4}{5}x_5$; min LB = 0

(d) $x_4 \geq 2 - x_1 + 3x_2 - 5x_3 - x_4 + 4x_5$; min LB = -5

(e) $x_5 \leq -\frac{2}{4} + \frac{1}{4}x_1 - \frac{3}{4}x_2 + \frac{5}{4}x_3 + \frac{1}{4}x_4$; max UB = $\frac{5}{4}$

Thus we do not learn anything we did not already know from Constraint 1.

Constraint 2

(a) $x_1 \leq 3x_2 - \frac{3}{2}x_3 - x_4 + x_5$; max UB = 4

(b) $x_2 \geq \frac{1}{3}x_1 + \frac{1}{2}x_3 + \frac{1}{3}x_4 - \frac{1}{3}x_5$; min LB = $-\frac{1}{3}$

(c) $x_3 \leq -\frac{2}{3}x_1 + 2x_2 - \frac{2}{3}x_4 + \frac{2}{3}x_5$; max UB = $\frac{8}{3}$

(d) $x_4 \leq -x_1 + 3x_2 - \frac{3}{2}x_3 + x_5$; max UB = 4

(e) $x_5 \geq x_1 - 3x_2 + \frac{3}{2}x_3 + x_4$; min LB = -3

Thus we learn nothing of value from Constraint 2.

Constraint 3

(a) $x_2 \leq -1 + 2x_3 - x_4 - x_5$; max UB = 1

(b) $x_3 \geq \frac{1}{2} + \frac{1}{2}x_2 + \frac{1}{2}x_4 + \frac{1}{2}x_5$; min LB = $\frac{1}{2}$

(c) $x_4 \leq -1 - x_2 + 2x_3 - x_5$; max UB = 1

(d) $x_5 \leq -1 - x_2 + 2x_3 - x_4$; max UB = 1

From (b), we learn that $x_3 = 1$. Now by eliminating x_3 and repeating the process, we could find final solution values for other variables and eventually solve the problem (McMillan [11]).

Clearly this becomes a tedious process. If we are to solve larger problems with many variables and many constraints we will need a superior search procedure. Balas' algorithm is such a procedure. Before examining Balas' algorithm we will discuss how to modify the 0-1 problem so that all coefficients in the objective function are positive.

7.2 Negative Coefficients in the Objective Function

The previous example was the simplest case of all positive integer coefficients in the objective function. We will assume that in all 0-1 problems negative coefficients will be replaced as illustrated in the following example:

Example 7.2: $\min z = 10x_1 - 5x_2 + x_3$

$$\begin{aligned} \text{st} \quad x_1 + 2x_2 - x_3 &\geq 2 \\ 2x_1 + x_2 + x_3 &\leq 3 \\ x_i = 0 \text{ or } 1,\ i &= 1, 2, 3 \end{aligned}$$

To eliminate the negative coefficient of x_2 in the objective function, we define a new variable:

$$y_2 = 1 - x_2 \quad \text{or} \quad x_2 = 1 - y_2$$

and the objective function becomes

$$10x_1 - 5(1 - y_2) + 3x_3$$

$$= 10x_1 + 5y_2 + 3x_3 - 5$$

Hence, the new problem is

$$\begin{aligned} \min z &= 10x_1 + 5y_2 + 3x_3 \\ \text{st} \quad & x_1 - 2y_2 - x_3 \geq 0 \\ & 2x_1 - \ y_2 + x_3 \leq 2 \end{aligned}$$

From the value of y_2, we can find the solution value of x_2.

7.3 Balas' Additive Algorithm

Egon Balas' procedure for solving 0-1 problems was first presented in 1965. His algorithm, like branch and bound, enumerates all solutions either explicitly or implicitly.

Given a 0-1 problem in two or three variables, we can quickly enumerate the solutions (feasible and nonfeasible) as shown below:

2 variables	3 variables	
(0, 0)	(0, 0, 0)	(0, 1, 1)
(0, 1)	(0, 0, 1)	(1, 0, 1)
(1, 0)	(0, 1, 0)	(1, 1, 0)
(1, 1)	(1, 0, 0)	(1, 1, 1)

When four or more variables are involved, the procedure to be used in calculating the possible solutions becomes more complex. An orderly procedure for enumerating the solutions, given any number of variables, is suggested by the network for four variables in Figure 7-1.

Each node in the network of Figure 7-1 specifies a solution. The numbers associated with a node represent the subscripts of variables that have values of 1 in that solution. To construct the network one draws, from each node, one line for each variable that does not have a value of 1 at the origin node. Each line "adds," for the variable it represents, a value of 1 at the node at which it terminates. Thus, from the node numbered 2, representing the solution (0, 1, 0, 0), lines labeled 1, 3, and 4 are drawn. The line labeled 1 terminates at a node in whose solution $x_1 = 1$; etc.

The importance of implicit enumeration is apparent if one imagines a four variable 0-1 problem in which $x_4 =1$ is infeasible. Solving by enumeration, we could now ignore all solutions in which $x_4 = 1$; i.e., the solutions associated with nodes eliminated from the network, as shown in Figure 7-2.

Similarly, suppose we find that the solution $x_2 = 1$, $x_3 = 1$, all other variables = 0, is infeasible (but not necessarily optimal). We could now ignore all solutions represented by nodes connected from below to node 2,3. All nodes connected from below node 2,3 represent solutions in which additional variables other than x_2 and x_3 have been given values of 1. Since all coefficients in the objective function are positive, these

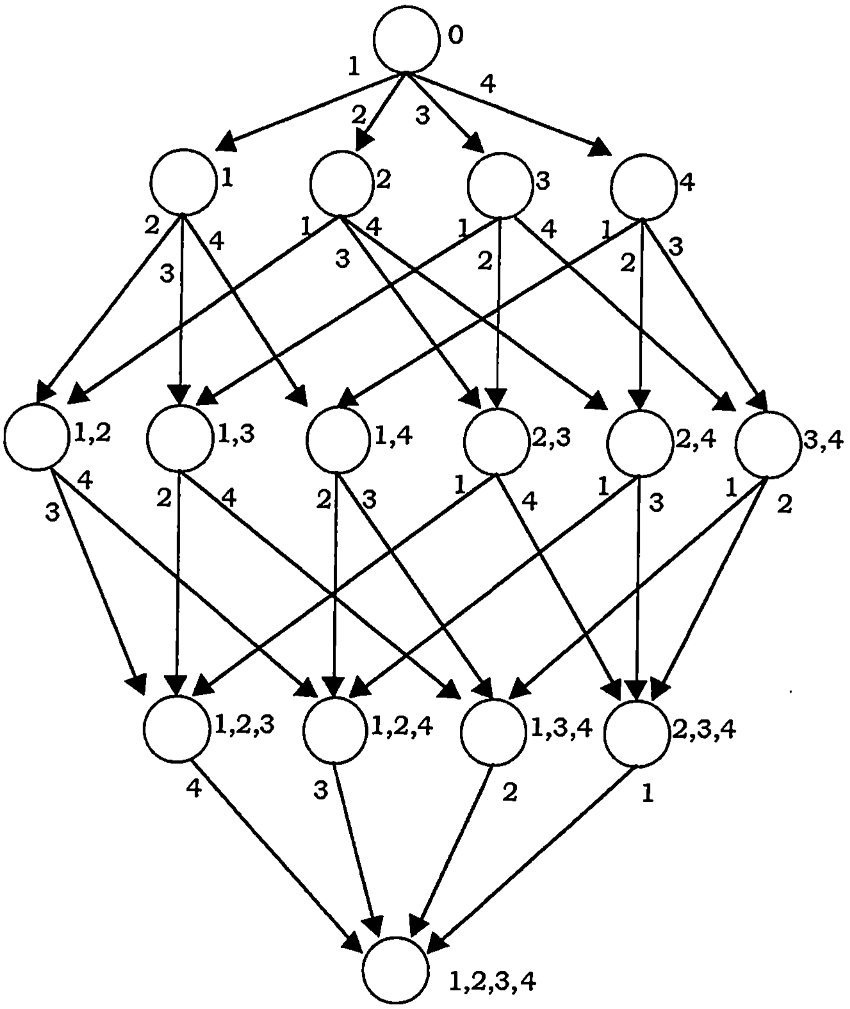


Figure 7-1. Network of solutions for a 4 variable 0-1 problem.

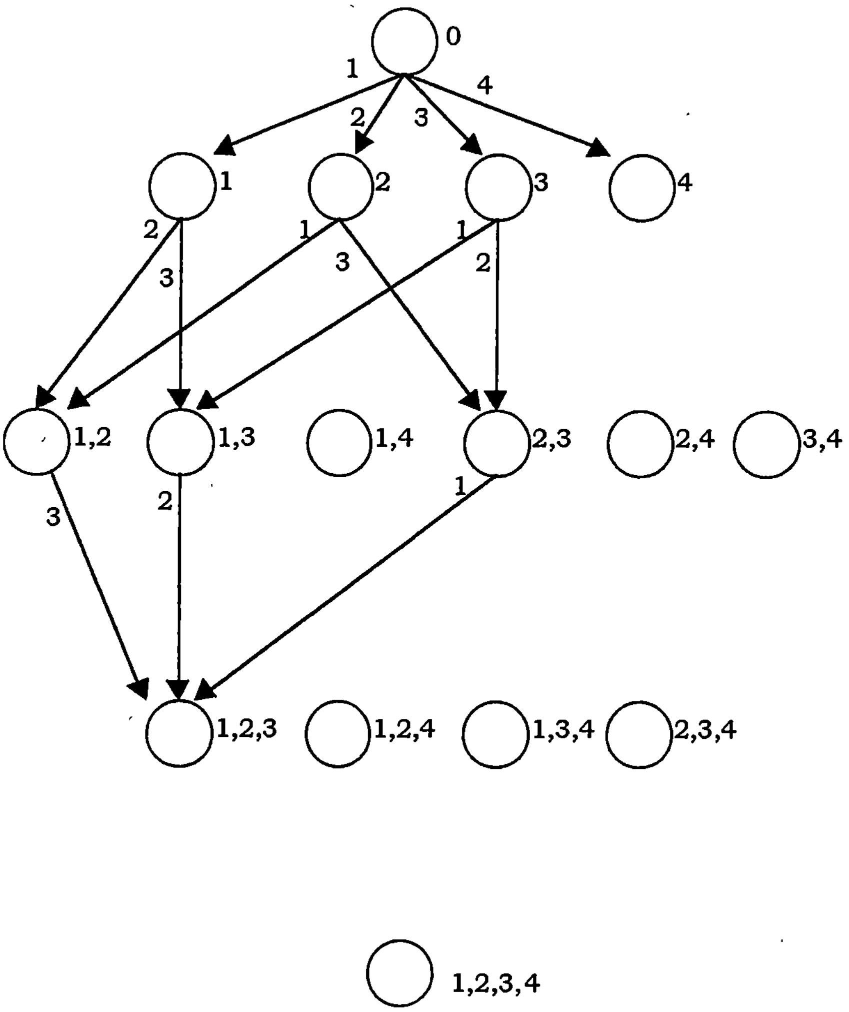


Figure 7-2. Network with solutions in which x_4 = 1 are eliminated.

solutions are less attractive. We can say they have been enumerated implicitly, and we can ignore them [11].

Before giving the procedure of the algorithm, we introduce some new terms and notation. Given a path P_k from node 0 to node k, the partition at node k is determined by choosing a free variable x_j and introducing

$$\left\{S^k \cap \{\mathbf{x}{:}x_j = 0\}, S^k \cap \{\mathbf{x}{:}x_j = 1\}\right\} \tag{7.3}$$

The path P_k corresponds to an assignment of binary variables to a subset of the variables. Such an assignment is called a *partial solution.* We denote the index set of assigned variables by $W_k \subseteq N$ and let

$$S_k^+ = \{j{:}\,j \in W_k \text{ and } x_j = 1\}$$

$$S_k^- = \{j{:}\,j \in W_k \text{ and } x_j = 0\}$$

$$S_k^0 = \{j{:}\,j \notin W_k\}$$

A completion of W_k is an assignment of binary variables to the free variables specified by the index set S_k^0.

Bounds

The problem considered at node k is

$$\min f(\mathbf{x}) = \sum_{j \in S_k^0} c_j x_j + \sum_{j \in S_k^+} c_j$$

$$\sum_{j \in S_k^0} a_{ij} x_j \leq b_i - \sum_{j \in S_k^+} a_{ij} = s_j, \; i = 1, \ldots, m \tag{7.4}$$

$$x_j = 0 \text{ or } 1,\ j \in S_k^0$$

Let $T^k = \{\mathbf{x}: x_j = 0 \text{ or } 1,\ j \in S_k^0\}$. Because $c_j \geq 0$, the relaxed solution $\mathbf{x}^k$ is obtained by setting $x_j = 0,\ j \in S_k^0$. Thus $\underline{z}^k = \sum_{j \in S_k^+} c_j$. Also, if $\mathbf{s} = (s_1, \ldots, s_m) \geq 0$, then $\mathbf{x}^k$ is feasible to (7.4) and $\overline{z}^k = z_{IP}^k = \sum_{j \in S_k^+} c_j$.

Fathoming

Node k can be fathomed if

a) $\underline{z}^k = \overline{z}^k$

b) $\underline{z}^k \geq \overline{z}^0$

using the bounds derived above. Note that (a) occurs when $\mathbf{x}^k$ is feasible to (7.4). A sufficient condition for (b) to hold can be easily evaluated. Suppose that for some i

$$t_i = \sum_{j \in S_k^0} \min \{0, a_{ij}\} > s_j$$

In this case, no completion of W_k can satisfy constraint i so $\underline{z}^k = \infty$, and node k is fathomed. For example, consider the constraint

$$\sum_{j \in S_k^0} a_{ij}x_j = -5x_1 + 4x_2 + 2x_3 - 3x_4 \leq -11 = s_i$$

The computations give $t_i = -8 > s_i = -11$ so node k is fathomed.

Partitioning and Branching

We would like to identify the subset of free variables at node k of which at least one must be equal to 1 in a feasible completion of W_k. Let $Q_k = \{ i\text{: } s_i < 0\}$. If $Q_k = \varnothing$, then node k is fathomed since $\mathbf{x}^k$ is feasible. If $Q_k \neq \varnothing$, let

$$R_k = \{j\text{: } j \in S_k^0 \text{ and } a_{ij} < 0 \text{ for some } i \in Q_k\}$$

At least one variable whose index is an element of R_k must equal 1 in any feasible completion of W_k. We can partition on some x_j, $j \in R_k$, and then branch to the successor node corresponding to $x_j = 1$. The following rule chooses such a $j \in R_k$ in an attempt to move toward feasibility. Define

$$I_k = \sum_{i=1}^{m} \max \{0, -s_i\} = -\sum_{i \in Q_k} s_i$$

to be the infeasibility of (7.4). By choosing x_j, the infeasibility at the successor node is

$$I_k(j) = \sum_{i=1}^{m} \max \{0, -s_i + a_{ij}\}$$

and x_p is selected such that

$$I_k(p) = \min_{j \in R_k} I_k(j)$$

For example, if the constraints at node k are

$$-6x_1 - 2x_2 + 2x_3 \leq -3$$

$$-3x_1 - 4x_2 + x_3 \leq -2$$

$$7x_1 + 5x_2 - 5x_3 \leq 4$$

then $R_k = \{1,2\}$, $I_k(1) = 3$, and $I_k(2) = 2$ so x_2 is chosen as the partitioning variable.

It is possible to represent P_k concisely in vector form. In Figure 7-3, P_9 is represented by the vector $(\underline{3},\ 1,\ \underline{4})$, where the order of the components indicates the level in the tree. Indices appear in the P_k vector if they are in W_k. They appear underlined if they are in $\overline{S_k}$. When branching to $x_p = 1$ from node k, we simply change P_k to (P_k, p). In Figure 7-3, branching to $x_2 = 1$ from node 9 yields $(\underline{3},\ 1,\ \underline{4},\ 2)$. In backtracking, we underline the rightmost nonunderlined entry and erase all entries to its right. In the figure, to backtrack from node 9 to node 7, the vector is transformed from $(\underline{3},\ 1,\ \underline{4})$ to $(\underline{3},\ \underline{1})$. When all entries in P_k are underlined and node k is fathomed, every element in W_k has been evaluated at 0 and 1, so the enumeration is complete.

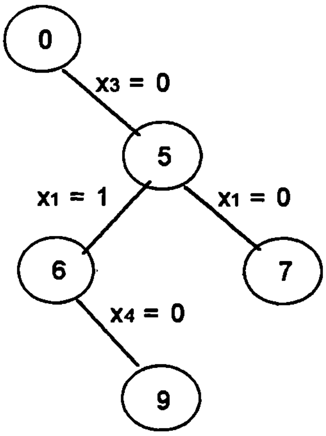


Figure 7-3. Example of a binary search tree.

Now we give the Balas additive algorithm which implements these ideas.

Step1 (Initialization) At node 0, $S_0^0 = \{1, \ldots, n\}$, $\underline{z}^0 = -\infty$ and $\overline{z}^0 = \infty$. Go to Step 2.

Step 2 (Calculating bounds) At node k, let $\underline{z}^k = \sum_{j \in S_k^+} c_j$. If $s_i \geq 0$ for all i, let $\overline{z^k} = z_{\text{IP}}^k = \underline{z}^k$ and let $\overline{z^0} = \min\left\{\overline{z^0}, \overline{z^k}\right\}$. Go to Step 3.

Step 3 (Fathoming) If $t_i > s_i$, for any i, or if $\underline{z}^k = \overline{z^k}$, or if $\underline{z}^k \geq \overline{z^0}$, fathom node k and go to Step 4. If node k is active, go to Step 5.

Step 4 (Backtracking) If no active node exists, go to Step 6. Otherwise, branch to the newest active node and go to Step 2.

Step 5 (Partitioning and branching) Partition on x_p as in (7.3), where $I_k(p) = \min_{j \in R_k} I_k(j)$. Branch in the direction of $x_p = 1$. Go to Step 2.

Step 6 (Termination) If $\overline{z^0} = \infty$, there is no feasible solution. If $\overline{z^0} < \infty$, the feasible solution that yielded $\overline{z^0}$ is optimal [1].

To illustrate the algorithm, we will look again at Example 7.1.

Example 7.1: $\min z = 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5$

$$\begin{aligned} \text{st} \quad x_1 - 3x_2 + 5x_3 + x_4 - 4x_5 &\geq 2 \\ 2x_1 - 6x_2 + 3x_3 + 2x_4 - 2x_5 &\leq 0 \\ - x_2 + 2x_3 - x_4 - x_5 &\geq 1 \end{aligned}$$

At Step 1, we have $\mathcal{S}_0^0$ = {1, ..., 5}, $\mathcal{S}_0^+$ = $\mathcal{S}_0^-$ = ∅, $\underline{z}^0 = -\infty$, $\overline{z}^0 = \infty$, **s** = (-2, 0, -1), I_0 = 3, and R_0 = {1, 3, 4}. Because all variables are free, we go to Step 5 and evaluate $I_0(1)$ = 4, $I_0(3)$ = 3, and $I_0(4)$ = 5. The minimum is associated with p = 3, so we select x_3 for partitioning and branch in the direction of x_3 = 1. The tree is shown if Figure 7-4.

At node 1, P_1 = (3), $\underline{z}^1 = -10$, **s** = (3, -3, 1), R_1 = {2, 5}, $I_1(2)$ = 0, and $I_1(5)$ = 2, so we choose x_2 for partitioning. No feasible solution is available yet so $\overline{z}^0 = \infty$. At node 2, P_2 = (3, 2), $\underline{z}^2 = 17$, and **s** = (0, 3, 0) so $\overline{z}^2 = 17$. We now put $\overline{z}^0 = \overline{z}^2 = 17$ and fathom node 2. At Step 4, we backtrack to node 3.

The computations continue at node 3, with P_3 = $(3, \underline{2})$, **s** = (3, -3, 1), and R_3 = {5}. But t_2 = -2 > s_2 = -3 so node 3 is fathomed, and we

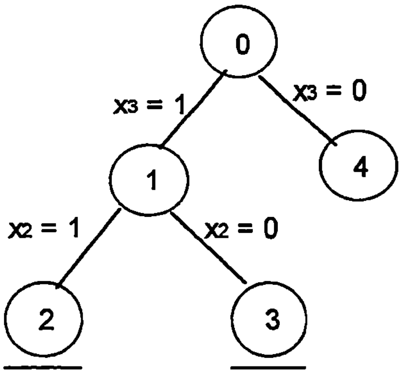


Figure 7-4. Search tree for Example 7.1.

backtrack to node 4. At node 4, $P_4 = (\underline{3})$, $s = (-2, 0, -1)$ and $R_4 = \{1, 4\}$. But $t_3 = 0 > s_3 = -1$ so node 4 is fathomed. At this point there are no active nodes so we go to Step 6 and terminate. The optimal solution is $z_{\mathrm{IP}} = 17$, $x^* = (0, 1, 1, 0, 0)$.

Among the advantages of Balas' algorithm are the following:

- addition is the only arithmetic operation required (no systems of linear equations need be solved), thus eliminating roundoff problems;
- a feasible solution is usually in store if the calculations are stopped prior to natural termination by exhaustion;
- it is easy to monitor the rate of implicit enumeration as the calculations proceed, which enables informed stopping rules and opportunistic implementation;
- only minor modifications are necessary to handle a variety of nonlinear objective functions (Geoffrion [7]).

7.4 General Branching

The branching rules in the additive algorithm can easily be modified to make it more general. For example, the requirement of branching on $x_p = 1$ in Step 5 can be relaxed by altering the vector representation of P_k. If branching to $x_p = 0$ prior to considering the node where $x_p = 1$ is desired, the bookkeeping scheme has to be extended. In particular, if $j \in W_k$, let it appear in P_k as

$$\begin{cases} j & \text{if } j \in S_k^+ \text{ and } x_j = 0 \text{ has not been considered} \\ \underline{j} & \text{if } j \in S_k^+ \text{ and } x_j = 0 \text{ has been considered} \\ -j & \text{if } j \in S_k^- \text{ and } x_j = 1 \text{ has not been considered} \\ -\underline{j} & \text{if } j \in S_k^- \text{ and } x_j = 1 \text{ has been considered} \end{cases}$$

The vector P_k is updated as before except that in backtracking, after erasing the underlined entries on the right, the rightmost remaining entry is underlined and its sign is changed. For example, if the order of nodes considered in Figure 7-4 had been node 1, node 3, node 2, node 4, the sequence of vectors would have been (3), (3, -2), $(3, -\underline{2})$, $(-\underline{3})$ [1].

A flowchart for the algorithm using general branching is given in Figure 7-5 [7].

Theorem 7.2. The procedure of Figure 7-5 leads to a nonredundant sequence of partial trial solutions which terminates only when all 2^n solutions have been (implicitly) enumerated.

Remark 1: By "nonredundant," we mean that no completion of a partial solution in the sequence ever duplicates a completion of a previous partial solution that was fathomed.

Remark 2: The proof below involves induction on the sequence $\langle P_k \rangle$ of partial solutions.

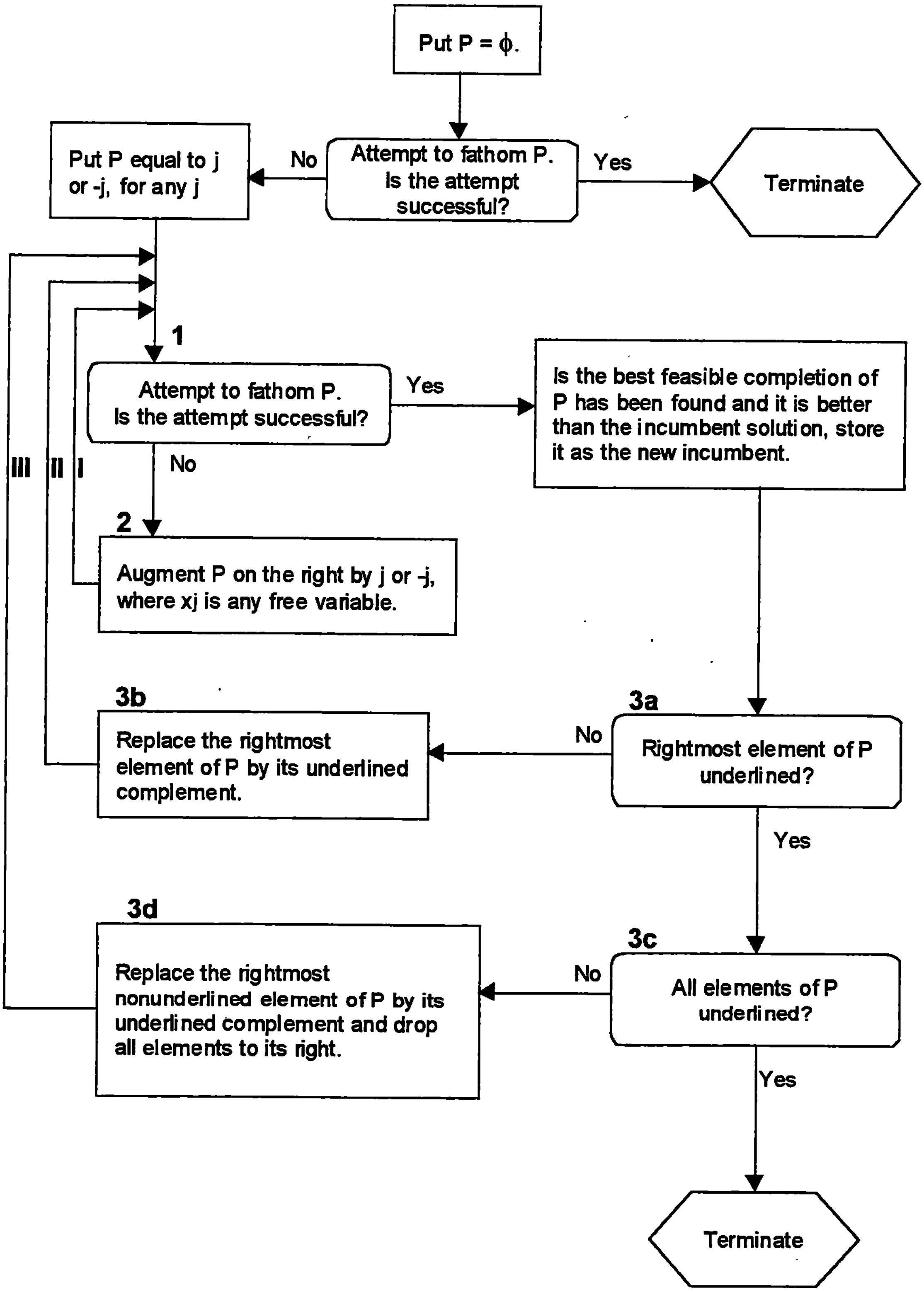

Put P = ϕ.
Attempt to fathom P. Is the attempt successful?
No
Yes
Terminate
Put P equal to j or -j, for any j
III
II
I
1
Attempt to fathom P. Is the attempt successful?
Yes
No
Is the best feasible completion of P has been found and it is better than the incumbent solution, store it as the new incumbent.
2
Augment P on the right by j or -j, where xj is any free variable.
3b
Replace the rightmost element of P by its underlined complement.
3a
No
Rightmost element of P underlined?
Yes
3d
Replace the rightmost nonunderlined element of P by its underlined complement and drop all elements to its right.
3c
No
All elements of P underlined?
Yes
Terminate

Figure 7-5. Flowchart for additive algorithm with general branching.

Proof: If Po = $\varnothing$ can be fathomed, the theorem is obviously true. Hence, we may assume that Po = $\varnothing$ cannot be fathomed.

To show that P_k is nonredundant, we will show that, if $P_1, ..., P_k$ are nonredundant, then P_{k+1} cannot be nonredundant; i.e., that P_{k+1} must include the complement of at least one element from each of the partial solutions fathomed prior to P_{k+1}. There are three pathways by which P_{k+1} can be determined from P_k; they are labeled I, II, III in Figure 7-5. If pathway I is taken, the desired conclusion follows from $P_k \subset P_{k+1}$. If pathway II is taken, the desired conclusion follows from the fact that P_k must have been determined by P_{k-1} by pathway I and hence $P_{k-1} \subset P_{k+1}$. To establish the desired conclusion for pathway III, we observe from Figure 7-5 that (i.) the element complemented in Step 3d was contained in every partial solution since it was originally introduced (and hence in every fathomed partial solution since that time), and that (ii.) P_{k+1} less its last element is not redundant with respect to the partial solution (if any) fathomed up to the time that the deleted element was introduced simply because P_{k+1} less its last element coincides with the actual partial solution at that time.

It remains to show that $<P_k>$ terminates only when all 2^n solutions have been (implicitly) enumerated. Clearly $<P_k>$ terminates only if a partial solution consisting of all underlined elements is fathomed. Our proof would be complete if we could show that every partial solution has the following property: *each underlined element implies that all completions of that portion up to and including the complement of the underlined element has been enumerated.* Now underlined elements have two possible origins: Steps 3b and 3d. Any underlined element created at Step 3b obviously has the asserted property. To see that the same is true of underlined elements created at Step 3d, consider the first time Step 3d is encountered. Then all underlined elements of the corresponding partial solution P must have been created at Step 3b; since P was just fathomed, therefore, by "telescoping" it follows that all completions of P less its rightmost consecutive underlined elements have been enumerated, i.e., this deleted partial solution has been fathomed. Thus, the new partial solution generated at the first execution of Step 3d has the desired property. A similar argument holds for each subsequent execution of 3d. The proof is now complete. ♦

CHAPTER 8

BRANCH-AND-CUT

8.1 Introduction

Recall that a *valid inequality* for a mixed IP is an inequality that is satisfied by all feasible solutions. Here we are interested in valid inequalities, called *cuts*, that are not part of the current formulation and are not satisfied by all feasible points to the LP relaxation. A *violated cut* is a cut that is not satisfied by the given optimal solution to the LP relaxation. We know from Chapter 5 that if we have a violated cut, we can add it to the LP relaxation and tighten it. By doing so, we modify the current formulation in such a way that the LP feasible region becomes smaller but the mixed IP feasible region does not change. Then we can resolve the LP and repeat the process, if necessary, so long as we can continue to find violated cuts [1].

The *branch-and-cut* method generalizes both pure cutting plane algorithms as well as branch-and-bound. Generally, at each node of the branch-and-bound tree, the LP relaxation is solved, violated cuts are found, these cuts are added to the relaxation, and the process is repeated. The goal is not only to reduce the number of nodes in the tree significantly by using cuts and improved formulations, but also by trying anything else that may be useful, such as preprocessing, at each node [19].

In practice there is a trade-off. If many cuts are added at each node, reoptimization may be slower than before. First, there is the additional time spent on trying to generate violated cuts, and this time is spent regardless of whether any violated cuts are found. Secondly, if one or more violated cuts are found, they are added to the active linear program which is then resolved. Therefore, we may be solving several linear programs per node. The linear programs become larger and are typically harder to solve. Consequently, we have to be careful and make sure that the extra time spent on evaluating nodes of the search tree does result in a smaller search tree and, more importantly, does result in a smaller overall solution time [1]. Thirdly, keeping all the information in the search tree is much more difficult. In branch-and-bound the problem to be solved at each node is obtained just by adding bounds. In branch-and-cut a cut pool is used in which all the cuts are stored. In addition to keeping the bounds and a good basis in the node list, it is also necessary to indicate which constraints are needed to reconstruct the formulation at the given node, so pointers to the appropriate constraints in the cut pool are kept [19].

8.2 Branch and Cut Algorithm

A branch-and-cut algorithm is outlined below. In the algorithm, L is the set of active nodes in the branch-and-cut tree. As in Chapter 6, the value of the incumbent solution is denoted by z^* and the value of the corresponding LP relaxation is denoted by z_{LP}. Also, z_{IP} is the value of the current IP optimum solution and z^i is the lower bound on the optimal value of the current subproblem under consideration.

Step 1 (Initialization): Denote the initial IP problem by IP^0 and set the active nodes to be $L = \{IP^0\}$. Set the upper bound to be $\bar{z} = +\infty$. Set $z^i = -\infty$ for the one problem $i \in L$.

Step 2 (Termination): If $L \neq \phi$, then the solution x^* which yielded the incumbent objective value z^* is optimal. If no such x^* exists, i.e., if $\bar{z} = +\infty$, then the IP is infeasible.

Step 3 (Problem selection): Select and delete a problem IP^i from L.

Step 4 (Relaxation): Solve the LP relaxation of IP^i. If the relaxation is infeasible, set $z^i = +\infty$ and go to Step 6. Let z^i denote the optimal objective value of the relaxation if it is feasible, and let x^i be an optimal solution. Otherwise, set $z^i = -\infty$.

Step 5 (Add cutting planes): If desired, search for cutting planes that are violated by x^i. If any are found, add them to the relaxation and return to Step 4.

Step 6 (Fathoming and Pruning):

a) If $z^i \geq z_{IP}$, go to Step 2.

b) If $z^i \leq z_{IP}$ and x^i is integral feasible, update $z_{IP} = z^i$, delete from L all problems with $z^i \geq z_{IP}$, and go to Step 2.

Step 7 (Partitioning): Choose a variable x_k with fractional part $f_k > 0$ and partition S^i as in (6.3). Go to Step 2.

The relaxations can be solved using any method for LP problems. Typically, the initial relaxation is solved using the simplex method. Subsequent relaxations are solved using the dual simplex method since the dual solution for the relaxation of the parent subproblem is still feasible in the relaxation of the child subproblem. Also, when cutting planes are added in Step 5, the current iterate is still dual feasible, so the modified relaxation can be solved using the dual simplex method (Mitchell [13]).

Example 8.1: We will solve the following problem using the branch and cut method:

$$\min z = -2x_1 - 5x_2$$

$$\begin{aligned} \text{st} \quad & 2x_1 - x_2 + x_3 = 9 \\ & 2x_1 + 8x_2 + x_4 = 31 \\ & x_j \geq 0, j = 1, \dots, 4 \\ & x_1, x_2 \text{ integer} \end{aligned}$$

To begin, the only active node is node 0 so we go to Step 4 and solve the corresponding LP. The optimal solution for the LP relaxation is shown in Tableau 8-1. Following the logic, we go to Step 7 and choose n x_2 for partitioning. Nodes 1 and 2 are created and are defined by the constraints $x_2 \le 2$ and $x_2 \ge 3$, respectively. We first solve the relaxation for Node 1 and obtain the optimal LP solution shown in Tableau 8-2.

We now go to Step 5. Since x_1 must be an integer, we choose row 1 to generate the cut and

$$\frac{1}{2}x_3 + \frac{1}{2}x_5 \ge \frac{1}{2} \quad \text{or} \quad x_3 + x_5 \ge 1$$

We now substitute for slack variables x_3 and x_5 to obtain

Tableau 8-1. Optimal LP solution for Example 8.1.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$
0	--	--	$-\frac{71}{3}$	0	0	$-\frac{1}{3}$	$-\frac{2}{3}$
1	$\mathbf{P}_1$	-2	$\frac{103}{18}$	1	0	$\frac{4}{9}$	$\frac{1}{18}$
2	$\mathbf{P}_2$	-5	$\frac{22}{9}$	0	1	$-\frac{1}{9}$	$\frac{1}{9}$

Tableau 8-2. Optimal LP solution for Node 1.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$
0	--	--	-21	0	0	-1	0	-6
1	$\mathbf{P}_1$	-2	$\frac{11}{2}$	1	0	$\frac{1}{2}$	0	$\frac{1}{2}$
2	$\mathbf{P}_4$	0	4	0	0	-1	1	-9
3	$\mathbf{P}_2$	-5	2	0	1	0	0	1

$$9 - 2x_1 + x_2 + 2 - x_2 \geq 1$$

or

$$x_1 \leq 5 \Rightarrow x_1 + x_6 = 6$$

We add this cutting plane constraint to the optimal LP tableau for Node 1, perform one dual simplex iteration, and obtain an optimal integer solution shown in Tableau 8-3.

We now backtrack and solve the LP relaxation for Node 2. The optimal LP solution is given in Tableau 8-4.

We need x_1 to be an integer, so we choose row 2 to generate the cut and

$$\frac{1}{2}x_4 + 4x_5 \geq \frac{1}{2} \quad \text{or} \quad x_4 + 8x_5 \geq 1$$

Tableau 8-3. Optimal integer solution for Node 1.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_6$
0	--	--	-20	0	0	0	0	-5	-2
1	$\mathbf{P}_1$	-2	5	1	0	0	0	0	1
2	$\mathbf{P}_4$	0	5	0	0	0	1	-8	-2
3	$\mathbf{P}_2$	-5	2	0	1	0	0	1	0
4	$\mathbf{P}_3$	0	1	0	0	1	0	1	-2

Tableau 8-4. Optimal LP solution for Node 2.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$
0	--	--	-22	0	0	0	-1	-3
1	$\mathbf{P}_3$	-2	5	0	0	1	-1	-9
2	$\mathbf{P}_1$	0	$7/2$	1	0	0	$1/2$	4
3	$\mathbf{P}_2$	-5	3	0	1	0	0	-1

Substituting for the slack variables x_4 and x_5 gives

$$31 - 2x_1 - 8x_2 + 8x_2 - 24 \geq 1$$

or

$$x_1 \leq 3 \Rightarrow x_1 + x_7 = 3$$

This cut is added to the relaxation for Node 2 and is solved using the dual simplex algorithm. The resulting solution is given in Tableau 8-5.

We now need x_2 to be an integer, so we choose row 3 of Tableau 8-5 to generate another cutting plane and

$$\frac{1}{8}x_4 + \frac{1}{36}x_7 \geq \frac{1}{8} \quad \text{or} \quad 9x_4 + 2x_7 \geq 9$$

We substitute for slack variables x_4 and x_7 as before and obtain

$$-20x_1 - 72x_2 \geq -276$$

or

$$5x_1 + 18x_2 \leq 69 \Rightarrow 5x_1 + 18x_2 + x_8 = 69$$

We add this cut to the tableau of the previous LP solution and solve using the dual simplex algorithm. The resulting solution is given by Tableau 8-6. The objective function value for Tableau 8-6, -21, is less than that of Tableau 8-3. Thus, Tableau 8-6 gives the optimal integer solution.

Figures 8-1 and 8-2 show the branch and cut tree before and after, respectively, cutting planes have been added. Notice in Figure 8-1 that further branching would be required if we were using a branch and bound method. This problem illustrates that using the branch and cut method can reduce the size of the search tree.

Tableau 8-5. LP solution after first cutting plane added.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_5$	$\mathbf{P}_7$
0	--	--	$-173/8$	0	0	0	$-5/8$	0	$-3/4$
1	$\mathbf{P}_3$	0	$49/8$	0	0	1	$1/8$	0	$-9/4$
2	$\mathbf{P}_1$	-2	3	1	0	0	0	0	1
3	$\mathbf{P}_2$	-5	$25/8$	0	1	0	$1/8$	0	$-1/4$
4	$\mathbf{P}_5$	0	$1/8$	0	0	0	$1/8$	1	$-1/4$

Tableau 8-6. Optimal integer solution for Node 2.

i	Basis	c	$\mathbf{P}_0$	$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_3$	$\mathbf{P}_4$	$\mathbf{P}_7$	$\mathbf{P}_8$
0	--	--	-21	0	0	0	0	$-11/18$	$-5/18$
1	$\mathbf{P}_3$	0	6	0	0	1	0	$-41/18$	$1/18$
2	$\mathbf{P}_1$	-2	3	1	0	0	0	1	0
3	$\mathbf{P}_2$	-5	3	0	1	0	0	$-5/18$	$1/18$
4	$\mathbf{P}_4$	0	1	0	0	0	1	$2/9$	$-4/9$

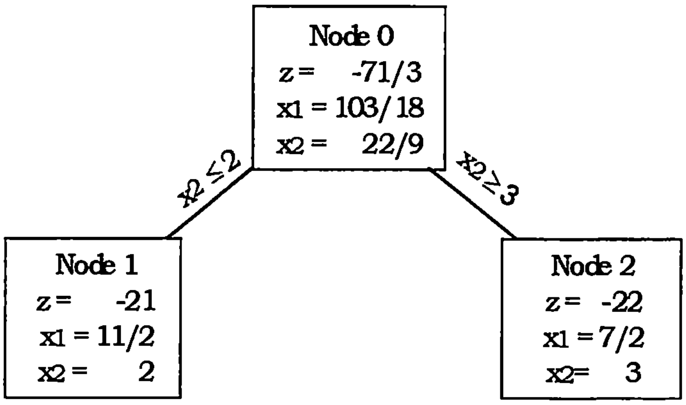


Figure 8-1: Branch-and-cut tree before cutting.

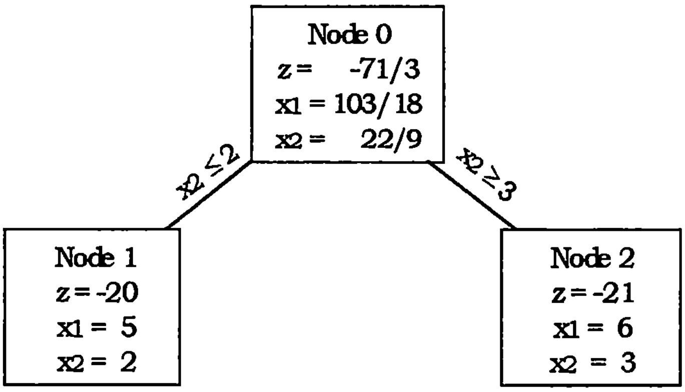


Figure 8-2: Branch-and-cut tree after cutting.

8.3 Cut Management

Cut management refers to strategies embedded in branch-and-cut algorithms to ensure effective and efficient use of cuts in an LP-based branch-and-bound algorithm. These strategies decide when to generate cuts, which of the generated cuts (if any) to add to the active linear program, and when to delete the previously generated cuts from the active linear program.

Several strategies have been investigated that try to decrease the time spent on generating cuts without reducing the effectiveness of the branch and cut algorithm. This can be done in two ways: (1) by limiting the number of times cuts are generated during the evaluation of a node, or (2) by not generating cuts at every node of the search tree.

The number of times violated cuts are generated during the evaluation of a node is often referred to as the *rounds of cut generation*. A popular strategy is to limit the rounds of cut generation to k_1 in the root node and k_2 in all other nodes with $k_1 > k_2$. Limiting the number of times violated cuts are generated during the evaluation of a node is not only important to reduce the time spent on cut generation, it also prevents tailing-off. *Tailing-off* refers to the phenomenon observed frequently that after several rounds of cut generation the objective function value hardly changes, i.e., even though violated cuts are identified and added to the active linear program, their addition does not lead to a substantial bound improvement. When tailing-off occurs, it may be beneficial to branch rather than generate cuts, since we may be spending significant amounts of time on cut generation without producing stronger bounds and thus without decreasing the size to the search tree.

Several strategies have been developed that generate violated cuts at only a subset of the nodes of the search tree. The simplest such strategy is to generate violated cuts at only the root node. The resulting branch-and-cut algorithm is called a *cut-and-branch* algorithm [1]. Usually an implementation of such a method will expend a great deal of effort on generating cutting planes, requiring much more time than just solving the relaxation at the root. The benefits of cut-and-branch include

- all generated cuts are valid throughout the tree since they are generated at the root;
- bookkeeping is reduced, since the relaxations are identical at each node;
- no time is spent generating cutting planes at other nodes

Cut-and-branch is an excellent technique for many general IP problems, but it lacks the power of branch-and-cut for some hard problems (Mitchell [12]).

A variant of this strategy generates cuts only in the top part of the search tree; i.e., at all nodes with depth less than or equal to some parameter *t*. The rationale behind these strategies is that generating cuts in the top part of the search tree is more important since it affects all nodes in the search tree. Another strategy is to generate violated cuts at every t^{th} node. The rationale behind this strategy is that after evaluating *t* nodes we have hopefully entered a different part of the search space and we will be able to generate violated cuts relevant for it.

As indicated above, another reason for the increase in time to evaluate nodes is that the addition of cuts leads to larger and more complex linear programs. To minimize the effects of these larger and more complex linear programs, a strategy is required that controls the size of the active linear program. The basic idea behind most control strategies is to have only a limited number of all available cuts in the active linear program. Since most of the cuts will not be binding in an optimal solution anyway, this seems reasonable. The cuts that are not part of the active linear program are kept in a so-called *cut pool* [1]. The pool of cuts makes it possible to reconstruct the parent node more efficiently, partly because difficulties with tailing-off are reduced [12].

Combining all the above ideas, we obtain the following basic cut management scheme:

1. Solve the active linear program.
2. Identify inactive cuts. If successful, delete them from the active linear program and move them to the cut pool.

3. Search cut pool for violated cuts. If successful, select a subset of them, add them to the active linear program, and go to 1.
4. Generate violated cuts. If successful, add them to the cut pool and go to 3.

Many implementation issues need to be addressed relating to the above scheme. First, we want to minimize the administrative overhead as much as possible. For example, we do not want to move cuts back and forth between the active linear program and the cut pool all the time. Secondly, we want to minimize the time it takes to search the cut pool for violated cuts. Therefore a cut pool typically has a fixed size. A small cut pool size obviously results in faster search times [1].

SECTION III: APPLICATION

CHAPTER 9

PLANT LOCATION PROBLEM

9.1 Fixed-Charge Problem

In a fixed-charge problem, there is a cost associated with performing an activity at a nonzero level. Problems in which a decision-maker must choose where to locate facilities are often fixed-charge problems. The decision-maker must choose where to locate various facilities (such as plants, warehouses, or business offices), and a fixed charge is often associated with building or operating a facility [18].

The fixed-charge problem defined as the mixed-integer problem

$$\min z = \sum_{j=1}^{n} (c_j x_j + k_j y_j) \tag{9.1}$$

$$\text{st} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i ,\ i = 1, \ldots, m \tag{9.2}$$

$$x_j \geq 0,\ j = 1, \ldots, n \tag{9.3}$$

$$y_j = \begin{cases} 0 & \text{if } x_j = 0 \\ 1 & \text{if } x_j > 0 \end{cases},\ j = 1, \ldots, n \tag{9.4}$$

involves the minimization of a concave objective function z over a convex polyhedron $Q = \{ x \mid \sum a_{ij} x_j \leq b_i ,\ i = 1, \ldots, m,\ x_j \geq 0, j = 1, \ldots, n\}$.

Constraints (9.2) and (9.3) are nothing more than the region defined in a linear programming problem [17]. To prove that the objective function is concave, we have the following theorem from Cooper and Drebes [2]:

Theorem 9.1. $f_j(x_j) = c_j x_j + k_j y_j,\ y_j = \begin{cases} 0 & \text{if } x_j = 0 \\ 1 & \text{if } x_j > 0 \end{cases}$

is a concave function for $x_j \geq 0$.

Proof: By definition, a function f(x) is concave if

$$f[\lambda x_1 + (1-\lambda)x_2] \geq \lambda f(x_1) + (1-\lambda)f(x_2),\ 0 \leq \lambda \leq 1.$$

Let $x_{j3} = \lambda x_{j1} + (1-\lambda)x_{j2}$. Then

$$f[\lambda x_{j1} + (1-\lambda)x_{j2}] = f_j(x_{j3}) = c_j x_{j3} + k_j y_{j3},$$

where $y_{j3} = \begin{cases} 0 & \text{if } x_{j3} = 0 \\ 1 & \text{if } x_{j3} > 0 \end{cases}$

Similarly: $f_j(x_{j1}) = c_j x_{j1} + k_j y_{j1}$

$f_j(x_{j2}) = c_j x_{j2} + k_j y_{j2}$

Therefore, we need to show that

$$f_j(x_{j3}) \geq \lambda\ f_j(x_{j1}) + (1-\lambda)\ f_j(x_{j2}).$$

Substituting we have

$$c_j[\lambda x_{j1} + (1-\lambda)x_{j2}] + k_j y_{j3} \geq \lambda c_j x_{j1} + \lambda k_j y_{j1} + (1-\lambda)c_j x_{j2} + (1-\lambda)k_j y_{j2}$$

$$\Rightarrow \lambda c_j x_{j1} + (1-\lambda)c_j x_{j2} + k_j y_{j3} \geq \lambda c_j x_{j1} + (1-\lambda)c_j x_{j2} + \lambda k_j y_{j1} + (1-\lambda)k_j y_{j2}$$

which reduces to

$$y_{j3} \geq \lambda y_{j1} + (1-\lambda)y_{j2}.$$

We now prove this is always true.

Case 1: Suppose $y_{j3} = 0$. This implies that

$$\lambda y_{j1} + (1-\lambda)y_{j2} = 0 \Rightarrow x_{j1} = x_{j2} = 0.$$

This implies that $y_{j1} = y_{j2} = 0$, and therefore, $y_{j3} = \lambda y_{j1} + (1-\lambda)y_{j2}$.

Case 2: Suppose $y_{j3} = 1$. This implies that

$$\lambda y_{j1} + (1-\lambda)y_{j2} > 0.$$

This implies that either (a.) $x_{j1} > 0, x_{j2} > 0$

(b.) $x_{j1} > 0, x_{j2} = 0$

(c.) $x_{j1} = 0, x_{j2} > 0$

(a.) $x_{j1} > 0, x_{j2} > 0 \Rightarrow y_{j1} = y_{j2} = 1$

$\Rightarrow y_{j3} = \lambda y_{j1} + (1-\lambda)y_{j2} = 1$ for $0 \leq \lambda \leq 1$.

(b.) $x_{j1} > 0, x_{j2} = 0 \Rightarrow y_{j1} = 1, y_{j2} = 0$

$\Rightarrow y_{j3} \geq \lambda y_{j1} + (1-\lambda)y_{j2}$ for $0 \leq \lambda \leq 1$.

(c.) $x_{j1} = 0, x_{j2} > 0 \Rightarrow y_{j1} = 0, y_{j2} = 1$

$\Rightarrow y_{j3} \geq \lambda y_{j1} + (1-\lambda)y_{j2}$ for $0 \leq \lambda \leq 1$. ♦

An important property of this type of problem, proven by Hirsch and Dantzig [8], is that the optimum solution must occur at an extreme point of the feasible space, that is, it must be associated with a feasible basic solution of Q. This result is not surprising since removing the fixed charge $k_j y_j$ from the objective function reduces the problem to a linear programming problem.

9.2 Plant Location Problem

Plant location problems are a variation of the fixed-charge problem. In the simplest case, m sources (or facility locations) produce a single commodity for n customers each with a demand for d_j units ($j = 1, ..., n$). If a particular source i is operating (or facility is built), it has a fixed cost $f_i \geq 0$ and a production capacity $M_i > 0$ associated with it. There is also a positive cost c_{ij} for shipping a unit from source i to customer j. The problem is to determine the location of the operating sources so that capacities are not exceeded and demands are met, all at a minimal total cost. All data are assumed to be integral.

To model this problem, we let x_{ij} be the amount shipped from source i to customer j and define y_i to be 1 if source i is operating and 0 if it is not. Then the IP model is

$$\min z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i \qquad (9.5)$$

$$\text{st} \qquad \sum_{i=1}^{m} x_{ij} = d_j, \; j = 1, ..., n \qquad (9.6)$$

$$\sum_{j=1}^{n} x_{ij} \leq M_i y_j, \; i = 1, ..., m \qquad (9.7)$$

$$x_{ij} \geq 0, \text{ all } i, j \qquad (9.8)$$

$$y_i = 0 \text{ or } 1 \qquad (9.9)$$

The objective function (9.5) is the total shipping cost $\sum_i \sum_j c_{ij}x_{ij}$ plus the total fixed cost $\sum_i f_i y_i$. Note that f_i contributes to the sum only when $y_i = 1$, or source *i* is operating. Constraints (9.6) guarantee that each customer's demand is met. Inequality (9.7) ensures that we do not ship from a source which is not operating (M_i is the upper bound on the amount that may be shipped from source *i*) and it also restricts production from exceeding capacity. Problems of this form are referred to as *capacitated plant location problems* (CPLP) [15].

The model can be further complicated by assuming that the term $\sum_i \sum_j c_{ij}x_{ij}$ in the objective function is replaced by some nonlinear function (mostly concave). It can also be further simplified by relaxing the capacity restriction on the sources [17]. When the plants are uncapacitated, the problem is often referred to as a *simple plant location problem* (SPLP) [15].

As an example, we will consider the following problem from Winston [18]:

Example 9.1: A company is considering opening warehouses in four cities: New York, Los Angeles, Chicago, and Atlanta. Each warehouse can ship 100 units per week. The weekly fixed cost of keeping each warehouse open is $400 for New York, $500 for Los Angeles, $300 for Chicago, and $150 for Atlanta. Region 1 of the country requires 80 units per week, region 2 requires 70 units per week, and region 3 requires 40 units per week. The costs (including production and shipping costs) of sending one unit from a plant to a region are shown in Figure 9-1. We want to meet weekly demands at minimum cost, subject to the preceding information and the following restrictions:

1) If the New York warehouse is opened, then the Los Angeles warehouse must be opened.

2) At most two warehouses can be opened.

3) Either the Atlanta or the Los Angeles warehouse must be opened.

Solution: The company faces two decisions. First, the company must decide which plants to operate. We define for $i = 1, 2, 3, 4$,

$$y_i = \begin{cases} 1 & \text{if warehouse opened in city } i \\ 0 & \text{otherwise} \end{cases}$$

where $i = 1$ represents NY, $i = 2$ represents LA, $i = 3$ represents Chicago, and $i = 4$ represents Atlanta.

		To	
From	Region 1	Region 2	Region 3
New York	$20	$40	$50
Los Angeles	48	15	26
Chicago	26	35	18
Atlanta	24	50	35

Figure 9-1. Shipping costs from city *i* to region *j*.

The company must also decide which warehouses ship to which regions. We define x_{ij}, for $i = 1, 2, 3, 4$ and $j = 1, 2, 3$, to be the amount shipped from warehouse i to region j. This value must be greater than or equal to zero.

The company wants to minimize the weekly costs of meeting demand = cost of shipping + cost of operating warehouses. The variable cost of shipping from warehouse i to region j is incurred only if $x_{ij} \geq 0$. Therefore, the company's weekly costs of shipping are

$$20x_{11} + 40x_{12} + 50x_{13} + 48x_{21} + 15x_{22} + 26x_{23} + 26x_{31} + 35x_{32} + 18x_{33} + 24x_{41} + 50x_{42} + 35x_{43}.$$

The cost of operating warehouse i is incurred if and only if $y_i = 1$, so the weekly fixed cost is

$$400y_1 + 500y_2 + 300y_3 + 150y_4.$$

Thus, the company's objective function may be written as

$$\begin{aligned} \min z = {} & 20x_{11} + 40x_{12} + 50x_{13} + 48x_{21} + 15x_{22} \\ & + 26x_{23} + 26x_{31} + 35x_{32} + 18x_{33} + 24x_{41} + 50x_{42} \\ & + 35x_{43} + 400y_1 + 500y_2 + 300y_3 + 150y_4. \end{aligned} \tag{9.10}$$

From Figure 9-2 we can obtain the constraints needed in order to meet the demand without exceeding the capacity.

	Region 1	Region 2	Region 3	**Capacity**
New York	x_{11}	x_{12}	x_{13}	$100y_1$
Los Angeles	x_{21}	x_{22}	x_{23}	$100y_2$
Chicago	x_{31}	x_{32}	x_{33}	$100y_3$
Atlanta	x_{41}	x_{42}	x_{43}	$100y_4$
Demand	80	70	40	

Figure 9-2. Summary of demand and capacity.

The weekly demand can be modeled by the following constraints:

$$x_{11} + x_{21} + x_{31} + x_{41} = 80 \tag{9.11}$$

$$x_{12} + x_{22} + x_{32} + x_{42} = 70 \tag{9.12}$$

$$x_{13} + x_{23} + x_{33} + x_{43} = 40 \tag{9.13}$$

The warehouse capacities can be modeled by these inequalities:

$$x_{11} + x_{12} + x_{13} \leq 100y_1 \tag{9.14}$$

$$x_{21} + x_{22} + x_{23} \leq 100y_2 \tag{9.15}$$

$$x_{31} + x_{32} + x_{33} \leq 100y_3 \tag{9.16}$$

$$x_{41} + x_{42} + x_{43} \leq 100y_4 \tag{9.17}$$

The company faces three types of constraints:

Type 1 constraint: If the NY warehouse is opened, then the LA warehouse must also be opened.

Type 2 constraint: At most two warehouses can be opened.

Type 3 constraint: Either the Atlanta or the LA warehouse must be opened.

The type 1 constraint states that $y_1 = 1 \Rightarrow y_2 = 1$. We can accomplish this by including

$$y_1 \le y_2$$

$$\text{or } y_1 - y_2 \le 0 \tag{9.18}$$

The type 2 constraint says that at most two of y_1, y_2, y_3, y_4 can equal 1 and the others equal 0. To accomplish this we add

$$y_1 + y_2 + y_3 + y_4 \le 2 \tag{9.19}$$

The type 3 constraint states that $y_2 = 1$ or $y_4 = 1$. We accomplish this by adding the constraints

$$y_2 \ge p \Rightarrow y_2 - p \ge 0 \tag{9.20}$$

$$y_4 \ge 1 - p \Rightarrow y_4 + p \ge 1 \tag{9.21}$$

$$p = 0 \text{ or } 1 \tag{9.22}$$

To show that the last constraints are the ones we want:

$$p = 0 \Rightarrow y_2 = 0 \text{ or } 1 \Rightarrow y_4 = 1$$

$$p = 1 \Rightarrow y_2 = 1 \Rightarrow y_4 = 0 \text{ or } 1.$$

Then the problem the company needs to solve is:

$$\min z = 20x_{11} + 40x_{12} + 50x_{13} + 48x_{21} + 15x_{22} + 26x_{23} + 26x_{31} + 35x_{32} + 18x_{33} + 24x_{41} + 50x_{42} + 35x_{43} + 400y_1 + 500y_2 + 300y_3 + 150y_4$$

$$\text{st} \quad x_{11} + x_{21} + x_{31} + x_{41} = 80$$

$$x_{12} + x_{22} + x_{32} + x_{42} = 70$$

$$x_{13} + x_{23} + x_{33} + x_{43} = 40$$

$$x_{11} + x_{12} + x_{13} \le 100y_1$$

$$x_{21} + x_{22} + x_{23} \le 100y_2$$

$$x_{31} + x_{32} + x_{33} \le 100y_3$$

$x_{41} + x_{42} + x_{43} \leq 100y_4$

$y_1 - y_2 \leq 0$

$y_1 + y_2 + y_3 + y_4 \leq 2$

$y_2 - p \geq 0$

$y_4 + p \geq 1$

$p = 0$ or 1

$x_{ij} \geq 0$, $i = 1, 2, 3, 4; j = 1, 2, 3$

$y_i = 0$ or 1, $i = 1, 2, 3, 4$

To solve this problem we will use the LINDO software. The LINDO problem and solution windows for this problem and more information about LINDO can be found in the Appendix.) The solution obtained by LINDO tells us that the min z = $4750. The company should open warehouses in Los Angeles, which ships to regions 2 and 3, and in Atlanta, which ships to regions 1 and 3.

BIBLIOGRAPHY

BIBLIOGRAPHY

[**1**] Jonathan F. Bard, *Practical Bilevel Optimization: Algorithms and Applications*, Kluwer Academic Publishers, (1998).

[**2**] L. Cooper and C. Drebes, "An Approximate Solution Method for the Fixed Charge Problem", *Naval Research Logistics Quarterly,* Vol 14 (1967), 178 – 190.

[**3**] L. Cooper and S. Steinberg, *Methods and Applications of Linear Programming,* Saunders, (1974).

[**4**] George B. Dantzig, *Linear Programming and Extensions,* Springer-Verlag, (1963).

[**5**] ----------------------, "Reminiscences About the Origins of Linear Programming", *Proceedings of the International Congress on Mathematical Programming, Rio de-Janeiro, Brazil, April 6-8, 1981*, North – Holland (1984), 105 – 112.

[**6**] Saul Gass, *Linear Programming: Methods and Applications,* Boyd and Frasier Publishing, (1985).

[**7**] Arthur M. Geoffrion, "Integer Programming by Implicit Enumeration and Balas' Method", *SIAM Review,* Vol 9, Issue 2, (1967), 178 – 190.

[**8**] Warren Hirsch and George Dantzig, "The Fixed Charge Problem", *Naval Research Logistics Quarterly,* Vol 15, Issue 3, (1968), 413-424.

[**9**] Ellis Johnson, George Nemhauser, and Martin Savelsbergh, "Progress in Linear Programming Based Branch-and-Bound Algorithms: An Exposition", (1997). Submitted to *INFORMS Journal on Computing.* http://akula.isye.gatech.edu/~mwps/

[**10**] J. Linderoth, M.W.P. Savelsbergh, "A Computational Study of Search Strategies for Mixed Integer Programming", *INFORMS Journal on Computing,* Spring, (1999).

[**11**] Claude McMillan, Jr., *Mathematical Programming: An Introduction to the Design and Application of Optimal Decision Machines,* Wiley, (1970).

[**12**] John E. Mitchell, "Branch-and-Cut Algorithms for Combinatorial Optimization Problems". To appear in the *Handbook of Applied Optimization,* Oxford University Press, (2000). http://www.math.rpi.edu/~mitchj

[**13**] --------------------, "Branch-and-Cut Algorithms for Integer Programming", (1998). To appear in *Encyclopedia of Optimization.* http://www.math.rpi.edu/~mitchj/chrono.html

[**14**] A. Ravindran, Don Phillips, and James Solberg, *Operations Research: Principles and Practice,* Wiley, (1987).

[**15**] Harvey M. Salkin and Kamlesh Mathur, *Foundations of Integer Programming,* North-Holland, (1989).

[**16**] Linus Schrage, *Optimization Modeling with LINDO,* Duxbury Press, (1997).

[**17**] Hamdy A. Taha, *Integer Programming: Theory, Applications, and Computations,* Academic Press, (1975).

[**18**] Wayne L. Winston, *Operations Research: Applications and Algorithm,* Duxbury Press, (1994).

[**19**] Laurence A. Wolsey, *Integer Programming,* Wiley, (1998).

APPENDIX

APPENDIX

SOLVING IP PROBLEMS WITH LINDO AND WHAT'S BEST

A.1 LINDO

LINDO (Linear Interactive and Discrete Optimizer) was developed by Linus Schrage. It is a user-friendly computer package that can be used to solve linear, integer, and quadratic programming problems [18]. The main purpose of LINDO is to allow a user to quickly input a formulation, solve it, assess the correctness or appropriateness of the formulation based on the solution, and then quickly make minor modifications to the formulation and repeat the process.

A LINDO model has a minimum requirement of three things. It needs an objective, variables, and constraints. The objective function must always be at the start of the model and is initiated with either a MAX (for maximize) or MIN (for minimize). The end of the objective function and the beginning of the constraints is signified with any of the following:

subject to

such that

st

The end of the constraints is signified with the word END. The strict inequalities "<" and ">" are interpreted by LINDO as the loose inequalities "$\leq$" and "$\geq$".

Variables that are restricted to the values 0 or 1 are identified with the INTEGER or INT specification. General integers are identified in similar fashion by using GIN instead of INT. These variable specifications are entered after the END statement

To illustrate, the problem

max $3x_1 + x_2$

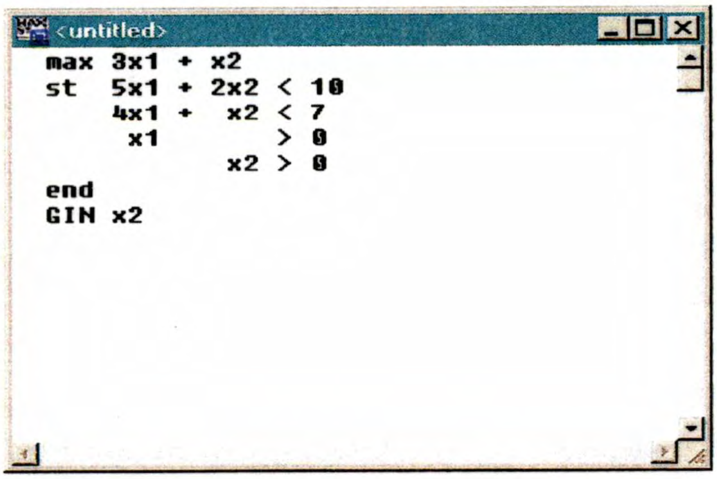
st $5x_1 + 2x_2 \leq 10$

$4x_1 + x_2 \leq 7$

$x_1, x_2 \geq 0$

x_2 integer

will appear in the model window as in Figure A-1.

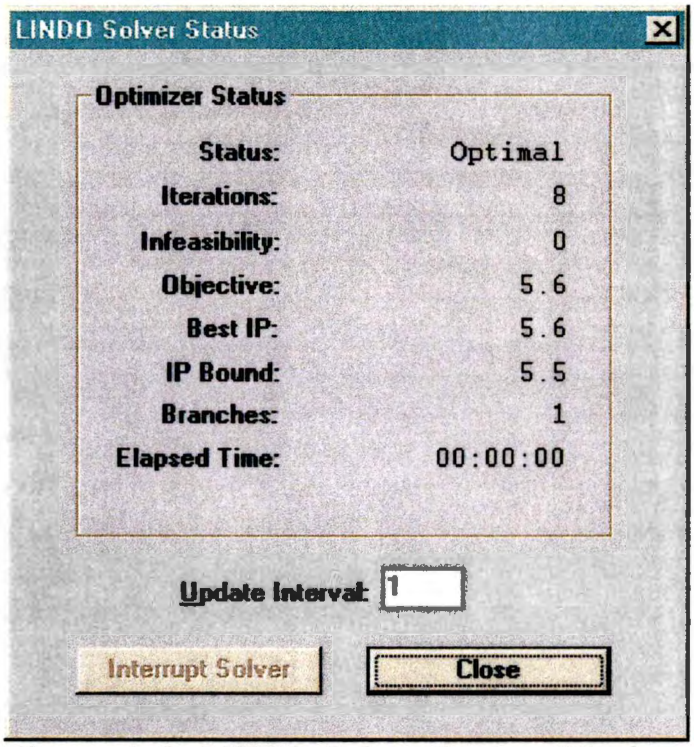


```
max 3x1 + x2
st  5x1 + 2x2 < 10
    4x1 +  x2 < 7
     x1       > 0
           x2 > 0
end
GIN x2
```

Figure A-1. LINDO model window.

The model is ready to solve. Click on the solve button, , or use the Solve command from the Solve menu. LINDO will begin by trying to compile the model. This means LINDO will determine whether the model makes mathematical sense and whether it conforms to syntactical requirements. If there are no formulation errors during the compilation phase, LINDO will begin to actually solve the model.

While solving, LINDO will show the Solver Status Window on your screen that looks like Figure A-2. This status window shows information about the model and the solution process. Below is a description of the various fields and controls within the Status window.



Status: Optimal
Iterations: 8
Infeasibility: 0
Objective: 5.6
Best IP: 5.6
IP Bound: 5.5
Branches: 1
Elapsed Time: 00:00:00

Update Interval: 1

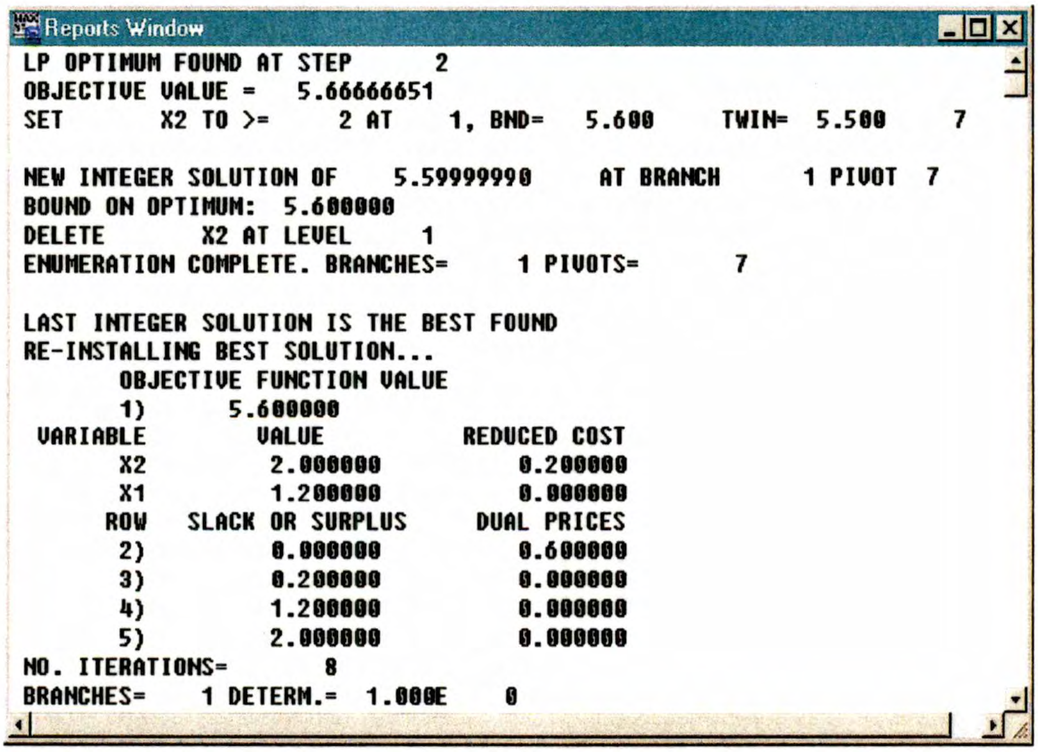
Figure A-2. LINDO Solver Status Window.

Field/Control	**Description**
Status	Gives status of current solution. Possible values include: Optimal, Feasible, Infeasible, Unbounded.
Iterations	Number of solver iterations.
Infeasibility	Amount by which constraints are violated.
Objective	Current value of the objective function.
Best IP	Objective value of the best integer solution found. Only relevant in IP models.
IP Bound	Theoretical bound on the objective for IP models.
Branches	Number of integer variables "branched" on by LINDO's IP solver.
Elapsed Time	Elapsed time since the Solver was invoked.
Update Interval	Frequency (in seconds) that the Status Window is updated. You can set this to any nonnegative value desired. Setting the interval to zero will tend to increase solution times.
Interrupt Solver	Interrupts the solver at any point. Returns the current best solution found.
Close	Closes the Status Window. Optimization continues. To reopen Status Window select the Status Window command from the Window menu.

At this point, LINDO will ask whether we wish to do sensitivity or range analysis. For our purposes here we answer "No" to this question. Then click on the "Close" button to close the Solver Status Window.

There will now be a new window titled "Reports Window." The Reports Window is where LINDO sends all text-based reporting output. This window contains the solution to our model and should resemble Figure A-3 [16].

In addition to the optimal solution, the LINDO output for an IP gives shadow prices and reduced costs. Unfortunately, the shadow prices and reduced costs refer to the subproblems generated during the



```
LP OPTIMUM FOUND AT STEP       2
OBJECTIVE VALUE =    5.66666651
SET        X2 TO >=      2 AT     1, BND=   5.600     TWIN=  5.500      7

NEW INTEGER SOLUTION OF     5.59999990      AT BRANCH       1 PIVOT   7
BOUND ON OPTIMUM:  5.600000
DELETE        X2 AT LEVEL      1
ENUMERATION COMPLETE. BRANCHES=      1 PIVOTS=        7

LAST INTEGER SOLUTION IS THE BEST FOUND
RE-INSTALLING BEST SOLUTION...
       OBJECTIVE FUNCTION VALUE
       1)       5.600000
 VARIABLE          VALUE          REDUCED COST
       X2          2.000000          0.200000
       X1          1.200000          0.000000
      ROW   SLACK OR SURPLUS      DUAL PRICES
       2)          0.000000          0.600000
       3)          0.200000          0.000000
       4)          1.200000          0.000000
       5)          2.000000          0.000000
NO. ITERATIONS=        8
BRANCHES=     1 DETERM.=  1.000E    0
```

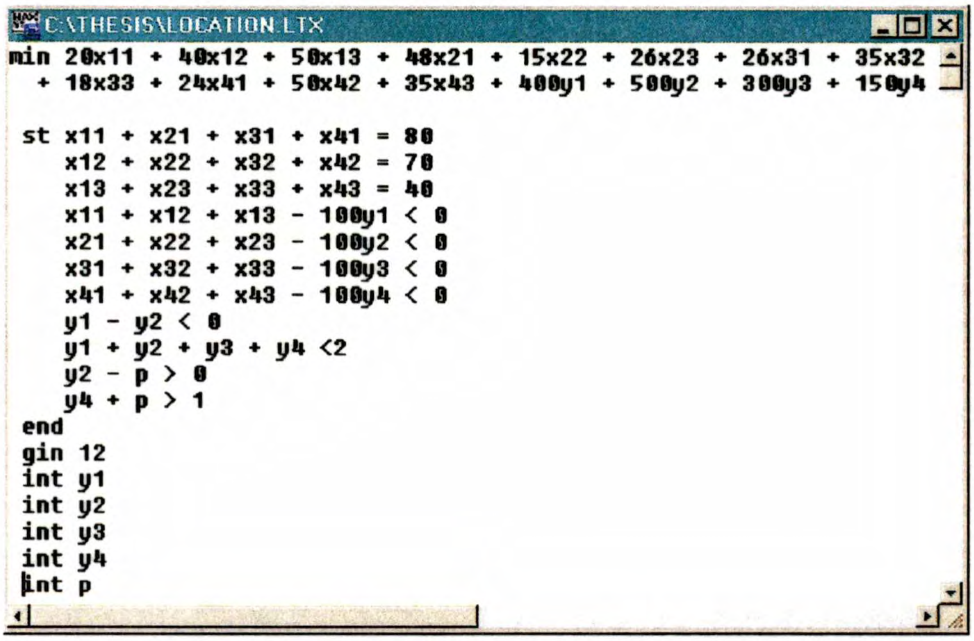
Figure A-3. LINDO Reports Window.

branch-and-bound solution – not to the IP. Associated with each variable in any solution is a quantity known as the *reduced cost.* The REDUCED COST portion of the LINDO output gives information about how changing the objective function coefficient for a nonbasic variable will change the optimal solution of the LP.

We define the *shadow price* for the i^{th} constraint to be the amount by which the optimal z-value is improved – decreased in a min problem and increased in a max problem – if the right-hand side of the i^{th} constraint is increased by 1. This definition applies only if the change in the right-hand side of constraint *i* leaves the current basis optimal. If, after a change in a constraint's right-hand side, the current basis is no longer optimal, the shadow prices of all constraints may change. The shadow prices for each constraint is found in the DUAL PRICES section of the LINDO output [18].

LINDO uses the convention that a positive dual price means that increasing the right-hand side in question will improve the objective function value. Similarly, a negative dual price means increasing the right-hand side will cause the objective function value to decrease. A zero dual price means that changing the right-hand side by a small amount will have no effect in the solution value. Therefore, under this convention, ≤ constraints will have nonnegative dual prices, ≥ constraints will have nonpositive dual prices, and = constraints can have dual prices of any sign [16].

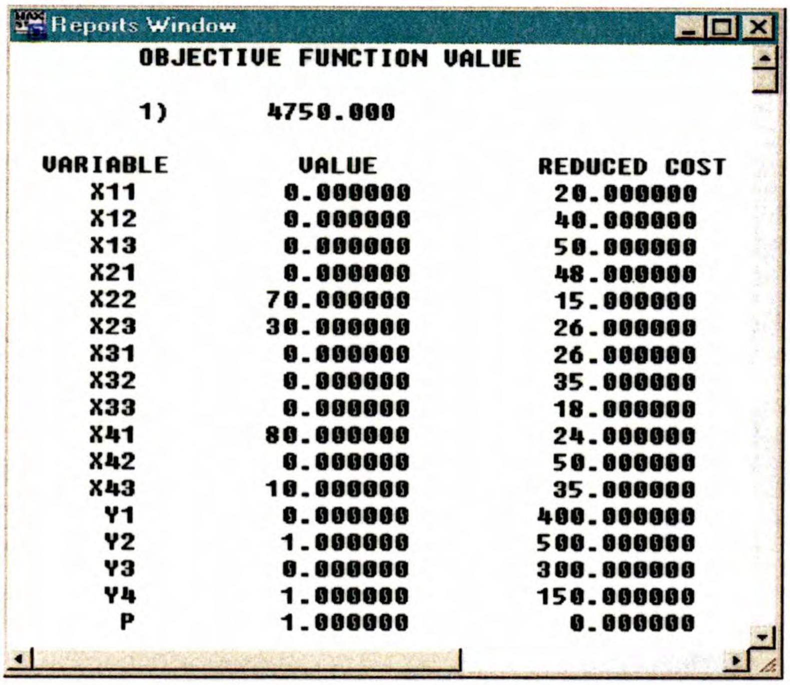
To show that LINDO can solve more complicated problems, we will use it to solve the plant location problem from Chapter 9. The model window for this problem is given in Figure A–4. The Reports Window showing the optimal solution and the Reduced Costs is given in Figure A-5.



```
min 20x11 + 40x12 + 50x13 + 48x21 + 15x22 + 26x23 + 26x31 + 35x32
  + 18x33 + 24x41 + 50x42 + 35x43 + 400y1 + 500y2 + 300y3 + 150y4

 st x11 + x21 + x31 + x41 = 80
    x12 + x22 + x32 + x42 = 70
    x13 + x23 + x33 + x43 = 40
    x11 + x12 + x13 - 100y1 < 0
    x21 + x22 + x23 - 100y2 < 0
    x31 + x32 + x33 - 100y3 < 0
    x41 + x42 + x43 - 100y4 < 0
    y1 - y2 < 0
    y1 + y2 + y3 + y4 <2
    y2 - p > 0
    y4 + p > 1
 end
 gin 12
 int y1
 int y2
 int y3
 int y4
 int p
```

Figure A-4. Model window for plant location problem.



```
          OBJECTIVE FUNCTION VALUE

          1)        4750.000

  VARIABLE        VALUE               REDUCED COST
       X11        0.000000            20.000000
       X12        0.000000            40.000000
       X13        0.000000            50.000000
       X21        0.000000            48.000000
       X22       70.000000            15.000000
       X23       30.000000            26.000000
       X31        0.000000            26.000000
       X32        0.000000            35.000000
       X33        0.000000            18.000000
       X41       80.000000            24.000000
       X42        0.000000            50.000000
       X43       10.000000            35.000000
        Y1        0.000000           400.000000
        Y2        1.000000           500.000000
        Y3        0.000000           300.000000
        Y4        1.000000           150.000000
         P        1.000000             0.000000
```

Figure A-5. Reports window for plant location problem.

A.2 What's Best

What's Best is an add-in that can be used to solve linear (and nonlinear) programming problems within a spreadsheet.

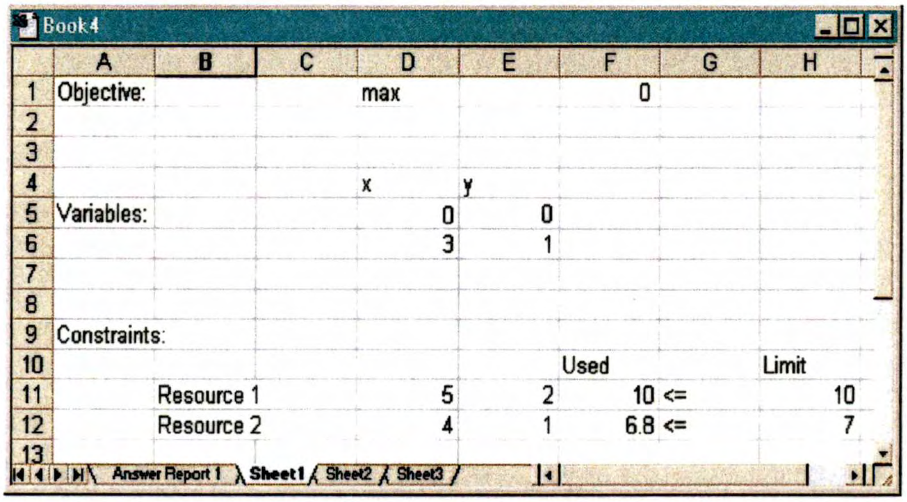
The key to solving an LP on a spreadsheet is to set up a spreadsheet that tracks everything of interest (costs or profits, resource usage, etc.). Then identify the cells in the spreadsheet that can be varied (these are called *adjustable* cells in What's Best). What's Best assumes (unless told otherwise) that all adjustable cells must be nonnegative. After defining the adjustable cells, identify the cell that contains the objective function (the *best* cell) and the constraint (the *constraint* cells). Then, select the Solve option to tell What's Best to solve the problem.

IPs can also be solved within a spreadsheet using What's Best. To define a range of variables as 0-1 variables, use the Binary option after defining the variables with the Adjust option. To define a range of variables to be general nonnegative integers, use the Integer option after defining the variables with the Adjust option [18].

To illustrate, we will look again at the example from the LINDO section, except that we will set $x_1 = x$ and $x_2 = y$:

$$\begin{aligned} &\max 3x + y \\ &\text{st}\quad 5x + 2y \leq 10 \\ &\qquad 4x + y \leq 7 \\ &\qquad x, y \geq 0 \\ &\qquad y \text{ integer} \end{aligned}$$

The associated spreadsheet looks like Figure A-6 after typing in the problem data. There are only a few formulas in the spreadsheet. The name xj has been defined as D5..E5. The name cj has been defined as D6..E6. The objective function value is in F1 and the formula is =SUMPRODUCT (cj,xj). The cells D11..E12 are just the numbers corresponding to those in the formulation. The formula in F11 is =SUMPRODUCT(D11..E11,xj), and this formula can be copied to cell F12. This sum counts the amount of the resource used. The cells H11..H12 are the right-hand side constants from the problem formulation.



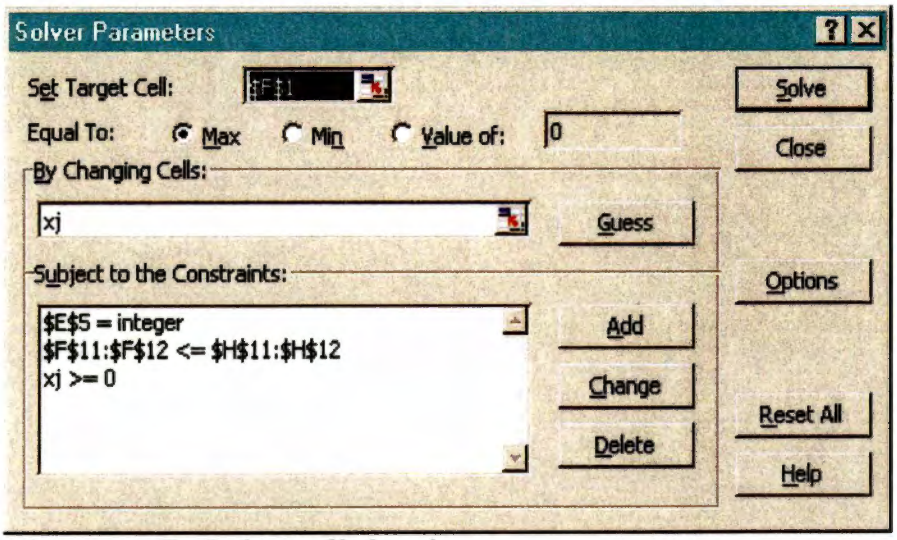
	A	B	C	D	E	F	G	H
1	Objective:			max		0		
2								
3								
4				x	y			
5	Variables:			0	0			
6				3	1			
7								
8								
9	Constraints:							
10						Used		Limit
11		Resource 1		5	2	10	<=	10
12		Resource 2		4	1	6.8	<=	7
13								

Figure A-6. What's Best model.

We are now ready to run the solver and find the optimal solution. Run the Solver option from the Tools menu, and you will see a dialog box. The Set Cell points to the cell that contains the objective function (F1 in this case). Click Max or Min as is appropriate to the problem (Max in this case). The window for By Changing Cells contains a list of the variable cells. Ours are named in a range window and we can just type the name xj.

To add constraints, click on the Add button and another dialog box appears. First we want to make sure x and y are positive, so under Cell Reference we type the range name, xj, select ≥ from the menu of operators and enter 0 in the Constraint box. Then click Add. To make y an integer, type the variable name, y, and select int from the menu of operators. The other two constraints are both ≤ so we can do them simultaneously. In the Cell Reference box we type the cells with the constraint SUMPRODUCT formulas, F11..F12 (we could also use the mouse to highlight these cells). Select ≤ and set the Constraint box to the right-hand side constants H11..H12. Click Add. Then click Cancel to dismiss this dialog box. Both constraints will show up in the main solver dialog box under Subject to the Constraints. To modify these constraints, highlight and click Change; to remove a constraint, highlight and click Delete.

Finally, we want to look at one more dialog box that contains the Options for the solver. This dialog box contains information to control the method the solver should use for the problem. For our purposes, make sure that Assume Linear Model is checked. Otherwise, the solver may terminate with a solution that is not optimal. Click OK. The Solver dialog box should now look like Figure A-7. Click Solve and a dialog box (Figure A-8) will appear. Make sure the Keep Solver Solution option is checked to have the spreadsheet report the answer the solver found. A box containing additional Reports appears on the right. Highlight Answer (Sensitivity and Limits do not apply for integer problems) and then click OK.



Set Target Cell: $F$1

Equal To: ● Max ○ Min ○ Value of: 0

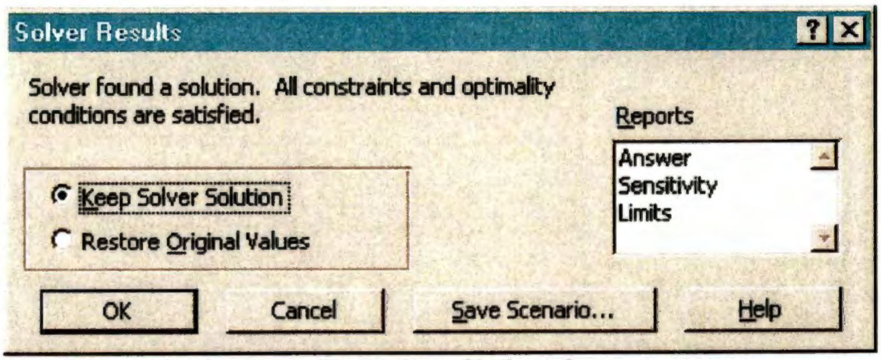
By Changing Cells:

xj

Subject to the Constraints:

$E$5 = integer
$F$11:$F$12 <= $H$11:$H$12
xj >= 0

Figure A-7. Solver dialog box.



Solver found a solution. All constraints and optimality conditions are satisfied.

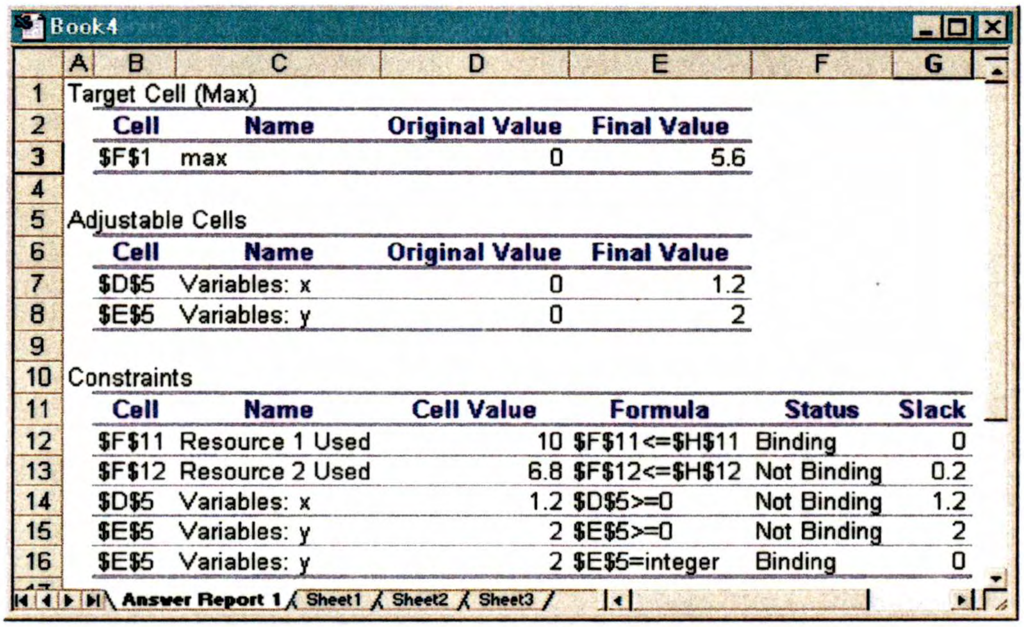
● Keep Solver Solution

○ Restore Original Values

Figure A-8. Solver Results dialog box.

The original spreadsheet will now show the optimal solution. In this case it shows that we should set x = 1.2 and y = 2 in order to maximize the objective function at a value of 5.6. Additionally, the Answer Report spreadsheet (Figure A-9) is created.

The Answer Report lists the objective function value, the decision variables, and the constraints. The reason the report shows Original



Target Cell (Max)

Cell	Name	Original Value	Final Value
$F$1	max	0	5.6

Adjustable Cells

Cell	Name	Original Value	Final Value
$D$5	Variables: x	0	1.2
$E$5	Variables: y	0	2

Constraints

Cell	Name	Cell Value	Formula	Status	Slack
$F$11	Resource 1 Used	10	$F$11<=$H$11	Binding	0
$F$12	Resource 2 Used	6.8	$F$12<=$H$12	Not Binding	0.2
$D$5	Variables: x	1.2	$D$5>=0	Not Binding	1.2
$E$5	Variables: y	2	$E$5>=0	Not Binding	2
$E$5	Variables: y	2	$E$5=integer	Binding	0

Figure A-9. Answer Report spreadsheet.

Value and the Final Value for the target cell is because if the solver is working on a nonlinear model, it will stop when time runs out or an improvement cannot be made. This is not necessarily optimal, so having the two values lets you see what improvements have been made.

For the Adjustable Cells and Constraints, Excel creates the Name fields by searching left of the cell for the first label, then up from the cell for the first label, then concatenates the two. By choosing those labels appropriately, the report can be made very descriptive. The labels can also be changed in the usual way since the report is just another spreadsheet. For each constraint, the report shows the formula that represents the constraint, whether it is binding (tight) or not binding (slack), and the amount of slack in that constraint.

VITA

Lisa Catherine Watkins was born in Greenville, South Carolina on July 13, 1971. She graduated from Shannon Forest Christian School in June, 1989. She then graduated from the University of South Carolina at Spartanburg in the Spring of 1998 with a Bachelor of Science in Mathematics and a Bachelor of Science in Secondary Education (Mathematics). While a student at USCS, Lisa was president of the Mathematics Club and was an active member of the Undergraduate Mathematics Research Group. She presented a paper at the Southeastern Section of the Mathematical Association of America meeting in Spring of 1997 and also presented at the Fourth Carolinas Mathematics Conference in the Fall of 1997. In April, 1997 she was inducted a member of the Kappa Delta Pi Honor Society in Education. She was inducted into the Gamma Beta Phi and the Omnicron Delta Kappa Societies in Spring of 1998. After accepting a Graduate Teaching Assistantship at The University of Tennessee, Knoxville, she entered the Graduate School there in August, 1998 in order to pursue the Master of Science degree in Mathematics. She graduated from The University of Tennessee in December of 1999 with a Master of Science in Mathematics with a concentration in Applied Mathematics.