

**An Application of Multiple Choice Programming to
Capital Budgeting Problems**

ARTHUR LAKES LIBRARY
COLORADO SCHOOL OF MINES
GOLDEN, CO 80401

By: Glen G. Roussos

ProQuest Number: 10794083

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10794083

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

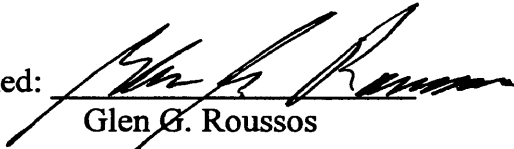
T-4671

A thesis submitted to the Faculty and Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date 2/3/95

Signed:



Glen G. Roussos

Approved:


Dr. Robert E. D. Woolsey

Golden, Colorado

Date 2/3/95


Dr. Graeme Fairweather
Professor and Head
Department of Mathematical
and Computer Sciences

ABSTRACT

There are, at the time of this study, two primary methods for the solution of integer programming (IP) problems. The first method uses branch and bound and the second method uses cutting planes. This thesis uses a linear programming based cutting plane algorithm to solve capital budgeting problems. The particular algorithm is called Multiple Choice Programming (MCP) and operates by deriving additional constraints from the final tableau of the linear programming (LP) formulation of the problem. By using the original constraints and the additional constraints generated, one can run the problem sequentially through linear programming iterations until an acceptable stopping criterion is met. This criterion will then cause the problem to converge to the optimal solution. This thesis demonstrates a reformulation method using “dummy” variables so that capital budgeting problems can conform to the requirements of MCP. Once these requirements are met, MCP can be used as a viable method of solving this specific class of integer programming problems.

The requirements of an MCP formulation are unique because this formulation must have integer sum constraints that sum to one. Capital budgeting problems typically do not have integer sum constraints so they must be added to the problems in order to use the MCP method. The additional constraints are created by a profit gradient which is derived from the reduced cost associated with negative values found in the final LP tableau. These additional constraints are recursively changed with each LP iteration; this finally drives the solution to optimal 0-1 values.

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vi
ACKNOWLEDGMENTS	vii
Chapter 1. INTRODUCTION	1
1.1 Thesis Topic	1
1.2 Overview of Capital Budgeting Problems	2
1.3 Theory and Practice	4
1.4 What is Multiple Choice Programming?	5
1.5 Motivation Behind MCP	6
1.6 Purpose of Thesis	6
Chapter 2. LITERATURE REVIEW	8
2.1 Previous Work in MCP	8
2.2 Integer Programming Methods	9
2.3 Cutting Plane Methods	12
2.3.1 General Discussion	12
2.3.2 The Fractional Algorithm	13
2.4 Branch and Bound Methods	18
2.4.1 General Discussion	18
2.4.2 Balas Additive Algorithm	22
Chapter 3. MULTIPLE CHOICE PROGRAMMING	27
3.1 General Discussion	27
3.2 Description of the Algorithm	29
3.3 Reformulation of Capital Budgeting Problems	32
3.4 Detailed Description	35
3.4.1 Algorithm to Solve Capital Budgeting Problems	39
3.5 Example Problems	44
3.5.1 Example Problem 1	45
3.5.2 Example Problem 2	53
3.6 Proof for the Profit Constraint	59

Chapter 4. RESULTS OF TEST PROBLEMS	62
4.1 Discussion	62
4.2 General Results	65
Chapter 5. CONCLUSIONS AND AREAS FOR FURTHER RESEARCH	70
5.1 Conclusions	70
5.2 Author's Contribution	71
5.3 Areas for Further Research	71
REFERENCES	75
Appendix A: Test Problems	78
Appendix B: MCP Code	92

LIST OF FIGURES

	<u>Page</u>
Figure 2.1: Graphic View of LP and IP Solution Space	10
Figure 2.2: LP Final Tableau	14
Figure 2.3: Tableau After Fractional Cut	17
Figure 2.4: Branch and Bound Tree	21
Figure 3.1: LP Example Final Tableau	36
Figure 3.2: Algorithm Flow Chart	40
Figure 3.3: Example 1 Tableau MCP Trial (0)	46
Figure 3.4: Example 1 Tableau MCP Trial (1)	49
Figure 3.5: Solutions for Example 1	51
Figure 3.6: Example 2 Tableau MCP Trial (0)	54
Figure 3.7: Solutions for Example 2	57
Figure 4.1: Results of Test Problems	66

ACKNOWLEDGMENTS

I want to thank Dr. Robert E. D. Woolsey, Dr. Ruth A. Maurer, Professor William R. Astle, and Dr. Warren Spaulding for their continued support and instruction throughout my studies at the Colorado School of Mines. Each of you has influenced me in many ways and will not be forgotten. You have made a significant impact on me and assisted me in realizing my vision.

I also want to thank Dr. William C. Healy, Jr. for his time and devotion in helping me to understand the motivation of the MCP algorithm. Several times I thought I was bringing him out of retirement but his patience never sagged. The members of the guild also deserve much appreciation, especially Erdem Ince and Tony Bertapelle who assisted me in writing portions of the MCP code.

Lastly, one cannot get far without venting their frustration to someone, that someone is my wife Johnna. She always supported me and did not complain when I was in a bad mood. Without her love and support, the last two years would have been infinitely more difficult.

Chapter 1

INTRODUCTION

1.1 Thesis Topic:

Solving capital budgeting problems is of real world importance in today's fast paced society. There have been many approaches to solving these particular types of mathematical problems, especially in the area of Integer Programming. Multiple Choice Programming (MCP) is one of these methods. The method was developed by W. C. Healy, Jr. [1] in the early 1960's while he was employed by Ethyl Corporation Research Laboratories, Ferndale, Michigan. MCP was a method developed to assist companies in making decisions on specific projects to gain the most benefit with specified constraints. This method was not tailored specifically to capital budgeting problems because these were not in the required format to be solved by the MCP method. We would argue that by reformulating capital budgeting problems one can solve them via MCP. To our knowledge, no test of MCP in solving capital budgeting problems presently exists in the literature of Operations Research/Management Science. This thesis will describe in detail a reformulation procedure for such problems and apply it to the MCP algorithm.

1.2 Overview of Capital Budgeting Problems:

Maximizing wealth of stockholders is a primary objective of most companies today. Those companies that do not bring money in for their investors soon perish. Capital budgeting problems are those types of problems that include investment strategies and Research and Development projects. Optimization is the goal [7]. The optimization is constrained, in that the company has a limited amount of manpower, money, time, equipment, and many other scarce resources to consider. Hence an organization must select the projects which will allow it to achieve the goal of optimizing its return on investment. A typical capital budgeting problem will often appear as follows:

Corporation XYZ wants to select the right mix of projects to maximize its return on investment. Each project will bring in a predetermined amount of profit to the company if the project is selected. However, since there are constraints which cannot be exceeded, all of the projects may not be undertaken. For example, each project requires a specified amount of money to accomplish and each project requires man hours to accomplish it. If all projects were selected, the available money and many years could be exceeded. The goal is to maximize profit subject to these constraints.

Often these problems can be represented by linear equations and inequalities and can be stated in the following form:

$$\begin{aligned}
 &\text{Maximize} && z^* = \sum_{j=1}^n c_j x_j \\
 &\text{subject to:} && \sum_{j=1}^n a_{ij} x_j \leq b_i \quad \text{for } i=1,2,\dots,m ; \quad a_{ij} \geq 0 \quad \forall i, \forall j \\
 &&& x_j = 0 \text{ or } 1 \quad \text{for } j=1,2,\dots,n. \quad \text{(eq. 1)}
 \end{aligned}$$

Where:

- z^* = Return on investment
- c_j = Amount of profit returned for project j
- x_j = Project j (where x_j 's are 0 or 1)
- a_{ij} = Constraining amount of item i needed to accomplish project j
- b_i = Total amount of constraining item i available

Capital budgeting problems also have another limiting factor besides resource allocation; the variables must assume integer values of either zero or one. The projects that are selected receive a value of one and the projects that are not selected are given a value of zero [4]. Unlike linear programming, a project cannot be done in fractional portions. Examples of the variables are investment portfolios, equipment, locations, etc. Therefore capital budgeting problems fall under the umbrella of 0-1 Integer Programming problems [8].

1.3 Theory and Practice

In general, there is some theory that should be applied to capital budgeting problems to determine the maximum benefit given certain constraints. A four stage model for capital budgeting can be described as follows:

- (1) Identification of an investment opportunity by an organization,
- (2) Development of the investment opportunity into a specific proposal,
- (3) Systematic method of project selection,
- (4) Control and follow through including postaudit, to assess return on investment and forecast accuracy [9].

Each of these four stages has several subareas that are considered during that particular stage. The two most important subareas are risk assessment and discounted cash flow (DCF) which fall under stages 3 and 4. Most companies have their own method of risk assessment which they prefer to use. Regardless of the method, this must be done carefully; an error can cause financial hardships in the future. DCF is of particular interest to a company because it allows the analysis of future profits in current dollars.

Contrasting capital budgeting practices with theory reveals a number of differences. The most apparent is that of selection, which is paramount. Many projects are rejected in the preselection stages because the organization failed to carefully analyze the risk assessment and the DCF properly [2]. In contrast, many are not rejected for the same reason. There may have been misallocated costs, oversight of some components, and

other economic considerations. Finally, many projects are selected for noneconomic reasons; these include personalities and politics. We assert that MCP can be best used in this area (stage 3), the process of project selection.

1.4 What is Multiple Choice Programming?:

MCP is a mixed integer programming method applied to problems that must have integer variables, specifically 0-1, as solutions. Additionally, the integer variables are divided into sets such that the variables in each set sum to unity (one). These integer sum constraints force the variables in the solution to be limited to values between zero and one when the problem iterates through a linear programming (LP) solver. The integer sum constraints, in conjunction with additional profit constraints, subsequently force the variables to assume values of zero or one.

A unique requirement for MCP is that an additional constraint must be added to the problem for each integer sum constraint. These additional constraints operate by recursively modifying an LP problem so as to drive successive LP solutions to an optimum solution having zero-one variables as required [1]. The method behind MCP relies on tightening a profit gradient (a partial derivative with respect to a variable). This will be described in detail in Chapter 3.

1.5 Motivation Behind MCP:

MCP addresses a variety of applications. It can be used to solve oil refinery problems, specifically pooling blending components and prohibiting reblending of split materials. W. C. Healy Jr. originally created MCP to address problems in production of oil based products [10]. He then found uses for it in mixed integer programming problems and fixed charge problems. Mixed integer programming problems are those types of problems that include both integer and continuous variables in the solution set. Fixed charge problems include a fixed cost for selecting a particular variable or solution.

1.6 Purpose of Thesis:

The purpose of this thesis is to develop a method for reformulating capital budgeting problems so that they can be solved by MCP. This will be done by the use of a matrix generator which automatically will reformulate the original problem created by the user into the MCP format. Subsequently the user can run this problem on the MCP code that is written in QBASIC. I will demonstrate this method by running several test problems and compare the solutions to that of the known optimal solutions. My goal is to expand the realm of MCP. The main justification for this work is the well known, and well documented, propensity of integer programming problems of very small size (less than 20 variables) to run for thousands of iterations before convergence [23]. As MCP does not rely on present integer programming theory we would propose that, with

sufficient testing, MCP may prove to be a viable alternative to presently used uneconomic integer programming algorithms.

Chapter 2

LITERATURE REVIEW

2.1 Previous Work in MCP

W. C. Healy Jr. developed MCP in the early 1960's when Integer Programming was in its infancy. Many people were trying various techniques of solving integer programming problems. The two main methods proposed, at that time, were that of branch and bound and cutting planes. MCP falls into the area of cutting planes. Shortly after the publication of his article, W. C. Healy Jr. left Ethyl Corporation Research Laboratories and made a transition into another job that took him away from the research arena [11]. Subsequently, many other integer programming methods emerged which had more apparent utility than MCP and Healy's work. Additionally, his work may have been ignored by theoreticians because there is no written proof that an optimal solution can be obtained.

The only other work done in the area of MCP was a Ph.D. dissertation done by Mark Woempner which addressed a question of degeneracy [12]. Degeneracy is encountered when alternate optima exists in a linear programming (LP) problem. It is caused when two variables in the basis have the same value that meet the criteria for leaving the basis in linear programming. This criteria is taken from the final tableau of an

LP iteration. Degeneracy has the potential for creating a cycling of variables in and out of the basis, thus leading to computational difficulties in the simplex method of LP [13].

2.2 Integer Programming Methods:

Integer programming is a branch of mathematical programming in which some or all of the variables can only assume integer values as their solutions. Unlike LP, where very large problems can be successfully solved in a reasonable amount of time, the performance of integer programming algorithms has been erratic [4]. This has been partly due to the convergence of the algorithm used. Additionally, LP solutions can take on fractional values. Why, then, would someone go to the trouble of trying to get a suitable integer programming algorithm when you could use LP and then round the solution to an integer? For example, if the continuous optimum indicates the number of busses required in a transportation problem is 7.2, this can be approximated by rounding to 7. There is no guarantee, however, that the rounded solution is the optimal integer solution or satisfies the constraints. Consider the following example:

$$\begin{aligned} \text{Max. } z^* &= 3x_1 + 4x_2 \\ \text{s. t. } & 22x_1 + 23x_2 \leq 86 \\ z_{LP}^* &= 11.73 \\ x_{1LP} &= 43/11 \\ x_{2LP} &= 0 \end{aligned}$$

If we approached this problem by rounding down to the optimal LP solution we would then have:

$$\begin{aligned} z_{LP}^* &= 9 \\ x_{1LP} &= 3 \\ x_{2LP} &= 0 \end{aligned}$$

Graphically this problem can be seen as:

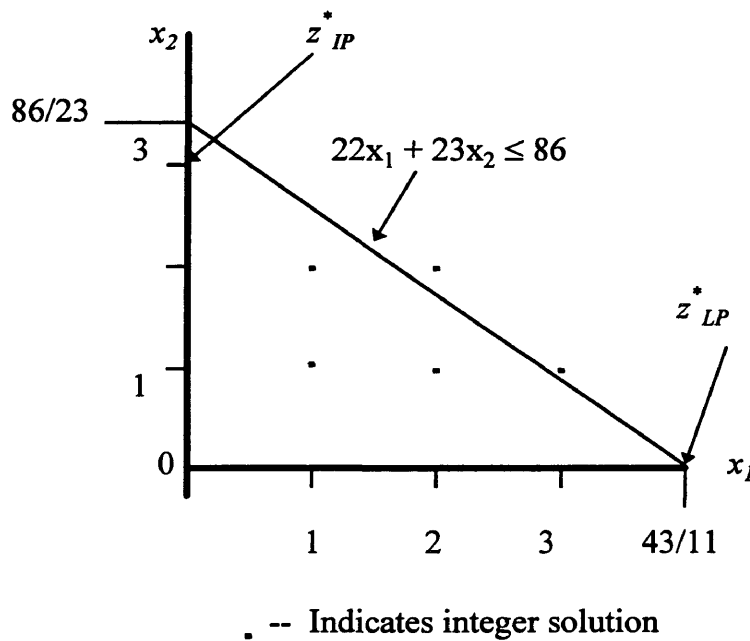


Figure 2.1 Graphic View of LP and IP Solution Space

The rounded down LP solution of 9, however, is not the optimal integer solution; the optimal integer solution is:

$$\begin{aligned} z_{IP}^* &= 12 \\ x_1 &= 0 \\ x_2 &= 3. \end{aligned}$$

Clearly, rounding down does not give the optimal integer solution in this case. For this reason, integer programming algorithms have been developed to find the optimal solutions to these types of problems.

Integer programming can be categorized into (1) cutting methods and (2) search methods. Cutting methods, also known as cutting planes, are primarily for integer linear problems that start with the continuous optimum. By systematically adding special constraints to the problem, the continuous solution space is gradually modified until its optimum extreme point satisfies the integer conditions. Thus the name “cutting plane” stems from the fact that the additional constraints force a “cut” from the solution space that does not contain integer points [4]. Search methods, also known as branch and bound and enumeration, search the feasible region and check all possible integer solutions for the optimal answer.

Integer programming is a relatively new field in the area of operations research. It was heavily researched in 1954 by Dantzig, Fulkerson, and Johnson through the use of cutting planes to turn linear programs into integer programs [14]. The most well known method, the Gomory cutting plane method, was introduced in 1958 by Gomory. He developed a systematic method for new constraint generation that is theoretically able to solve any integer linear problem [15]. Since then his work has been continued by several mathematicians, including Glover [16].

Following the initial work in the area of cutting planes, a new method evolved, that of implicit enumeration [4]. This was first considered by Dantzig in 1960 and later developed by Little, Murty, Sweeney, and Caroline in 1963 [17]. While this method is provably convergent to optimality, the computational time could be enormous depending on the size of the solution space. There have also been many strides made in the area of branch and bound, especially by Balas [18] and Lawler and Bell [19]. Presently, however, implicit enumeration methods are the most popular for solving integer programs. This is primarily due to today's technology in the area of computer processing capability. Typically, today's software packages will solve the initial linear program and then begin to enumerate within the solution space for the optimal answer. Two packages which use this approach are LINDO [21] and STORM [22].

2.3 Cutting Plane Methods:

2.3.1 General Discussion:

The use of cutting planes is one of the oldest methods used in solving integer problems [20]. Dantzig, Fulkerson, and Johnson first used the idea of cutting planes to solve "traveling salesman" problems (class of 0-1 integer problems) in 1954 [14]. The most well known, the Gomory cutting plane, was developed in 1958 and was used to solve all-integer linear programming problems [15]. Gomory extended his work into the

early 1960's to solve mixed integer problems. This ability to obtain the optimal integer solution slows dramatically as the number of variables increase in the problem [15].

The idea of the cutting plane algorithm is to change the convex solution space so that the optimal extreme point becomes all-integer. The change in the solution space should be made without cutting off any feasible integer solution of the original problem [4]. Consider the problem of the form previously proposed:

$$\begin{aligned} \text{Maximize} \quad & z^* = \sum_{j=1}^n c_j x_j \\ \text{subject to:} \quad & \sum_{j=1}^n a_{ij} x_j \leq b_i \quad \text{for } i=1,2,\dots,m ; \quad a_{ij} \geq 0 \quad \forall i, \forall j \\ & x_j \geq 0 \quad \text{for } j=1,2,\dots,n. \end{aligned} \quad (\text{eq. 1})$$

This problem is solved by isolating the optimal point with the required integer property in the convex solution space that is maximized. This optimal integer solution is found by generating additional constraints that continually “cut” the feasible region until the optimal n-space lattice point is a member of the newly formed feasible region.

2.3.2 The Fractional Algorithm:

The fractional algorithm is one particular algorithm of the cutting plane method. This is explained in detail by Taha [4]. The first requirement for this algorithm is that all of the coefficients and the right hand sides of the constraints must be integer. All

coefficients must be integers because fractional values in the coefficients may prevent the slack variables from assuming integer values. This would then indicate that no feasible integer solution exists, even though the problem may indeed have a feasible integer solution.

This is the general form of the algorithm. First, the problem is solved as a regular LP; that is, the solutions can assume fractional values. If the optimal solution happens to be all integer, then the problem terminates. Otherwise, the additional constraints added to the problem are developed as follows. Let the final tableau of the LP be given by:

Basic	x_1	\dots	x_i	\dots	x_m	w_1	\dots	w_j	\dots	w_n	Solution
z	0	\dots	0	\dots	0	c_1	\dots	c_j	\dots	c_n	β_0
x_1	1	\dots	0	\dots	0	α^1_1	\dots	α^j_1	\dots	α^n_1	β_1
.
.
.
x_i	0	\dots	1	\dots	0	α^1_i	\dots	α^j_i	\dots	α^n_i	β_i
.
.
.
x_m	0	\dots	0	\dots	1	α^1_m	\dots	α^j_m	\dots	α^n_m	β_m

Figure 2.2 LP Final Tableau

The variables x_i ($i = 1, 2, \dots, m$) are the basic variables and the variables w_j ($j = 1, 2, \dots, n$) are nonbasic variables.

Consider the i th equation where the basic variable x_i assumes a noninteger value.

Thus we have:

$$x_i = \beta_i - \sum_{j=1}^n \alpha_i^j w_j, \quad \beta_i = \text{noninteger} \quad (\text{eq. 2})$$

Such an equation in the basis is referred to as the source row. The source row will be used to make the additional constraints that eventually lead to a cut in the feasible region.

To attain the fractional cut, let:

$$\begin{aligned} \beta_i &= [\beta_i] + f_i \\ \alpha_i^j &= [\alpha_i^j] + f_{ij} \end{aligned} \quad (\text{eq. 3})$$

where $N = [a]$ is the largest integer such that $N \leq a$. Thus, it follows that $0 < f_i < 1$ and $0 \leq f_{ij} \leq 1$. This says that f_i can only be a positive fraction (between 0 and 1) and f_{ij} can only be a nonnegative fraction. Therefore the source row produces:

$$f_i - \sum_{j=1}^n f_{ij} w_j = x_i - [\beta_i] + \sum_{j=1}^n [\alpha_i^j] w_j \quad (\text{eq. 4})$$

In order for all the variables x_i and w_j to be integer, the right hand side of the equation must be integer; this then leads to the left side of the equation being integer. Since we have $f_{ij} \geq 0$ and $w_j \geq 0$ for all i and j , it follows that

$$\sum_{j=1}^n f_{ij} w_j \geq 0.$$

Furthermore, this leads to

$$f_i - \sum_{j=1}^n f_{ij} w_j \leq f_i \quad (\text{eq. 5})$$

However, equation 5 will always be less than 1 because $f_i < 1$. In order for the integer requirement to be satisfied, the left side must be integer, so equation 5 becomes

$$f_i - \sum_{j=1}^n f_{ij} w_j \leq 0 \quad (\text{eq. 6})$$

This last constraint can be put in the form

$$S_i = \sum_{j=1}^n f_{ij} w_j - f_i \quad (\text{eq. 7})$$

where S_i is a nonnegative slack variable which must be an integer. Equation 7 is the constraint that defines the fractional cut.

The new tableau after adding the fractional cut will then be

Basic	$x_1 \dots x_i \dots x_m$	$w_1 \dots w_j \dots w_n$	S_i	Solution
z	0 ... 0 ... 0	$c_1 \dots c_j \dots c_n$	0	β_0
x_1	1 ... 0 ... 0	$\alpha^1_1 \dots \alpha^j_1 \dots \alpha^n_1$	0	β_1
.
.
x_i	0 ... 0 ... 0	$\alpha^1_i \dots \alpha^j_i \dots \alpha^n_i$	0	β_i
.
.
x_m	0 ... 0 ... 0	$\alpha^1_m \dots \alpha^j_m \dots \alpha^n_m$	0	β_m
S_i	0 ... 0 ... 0	$-f_1 \dots -f_j \dots -f_m$	1	$-f_i$

Figure 2.3 Tableau After Fractional Cut

This tableau is presently infeasible because $S_i = -f_i$, which means the new constraint will not be satisfied by the given solution. The dual simplex method is then initiated to solve this problem; this is equivalent to cutting off a slice of the solution space toward the optimal integer solution. If the new solution, after applying the dual simplex method, is integer then the algorithm ends. Otherwise a new fractional cut is made on the new solution in the same manner as before. This process is repeated until an integer solution is obtained or the algorithm indicates no feasible integer solution exists.

This algorithm is called the fractional cut because all of the nonzero coefficients of the generated cuts are a value less than one. This algorithm can take a long time until it reaches the optimal integer solution because these cuts are so small; however, the cuts must ensure that no integer solution is being eliminated from the feasible region.

ARTHUR LAKES LIBRARY
 COLORADO SCHOOL OF MINES
 GOLDEN, CO 80401

Although other cutting methods may differ from this one, the rationale behind them is all the same: derive additional constraints that are added to the problem and force integer values to be members of the solution space.

2.4 Branch and Bound Methods:

2.4.1 General Discussion:

The branch and bound method is based on a combinatorial form of enumeration in a given solution space. One of the first branch and bound methods was presented in 1960 by Dantzig [23]. The method enumerated all possible integer combinations, but took an extremely long time to find the optimal solution. About the same time, Land and Doig designed their enumeration algorithm; many later algorithms have used their initial work as a foundation to make modifications. The term “branch and bound” actually came from the traveling salesman problem in 1963 by Little, Murty, Sweeney, and Karel [24]. In 1965, Dakin extended the work of Land and Doig with a simple variation. This work in turn was extended by Beale and Small to include linear post-optimization procedures [25]. Six years later a method which specialized the work of Land and Doig was introduced by Davis, Kendrick, and Weitzman. This approach was a method for solving zero-one integer problems; it incorporated the work of Driebeek [25] and branching strategies proposed by Spielberg [26]. Lastly, the most well known method of

enumeration is that of Balas. His work is primarily in the solution of binary (0-1) integer problems and mixed-integer problems [18].

The basic idea behind branch and bound is a sequential search procedure over all of the set of possible solutions to a 0-1 integer problem. This is done by dividing the sets into subsets. For each subset, upper and lower bounds are defined from the value of the objective function and the feasibility criteria to limit the search, ensuring that the number of possible solutions is finite. The property which is most important in branch and bound is its ability to enumerate the majority of the possible solutions so it can identify the optimal solution. The more solutions that can be enumerated implicitly, the faster the algorithm will find the optimal solution in the solution space. This is very important in integer programming because the number of possibilities of solutions grow at a rate of 2^n . For example, a problem with five binary (0-1) variables has $2^5 = 32$ possible solutions. Therefore a problem with 30 binary variables has $2^{30} = 1,073,741,824$ possible solutions. Obviously, these problems can involve lengthy calculations as the number of variables increase.

The implicit enumeration technique can best be visualized as a tree. Starting at the base of the tree, one must inspect each branch for possible solutions to the problem. After attaining these solutions, the constraints must be considered to ensure that none of them are being violated. If these constraints are violated the solution then becomes

infeasible. Consider the binary problem $2^3 = 8$ possible solutions, the complete enumeration can be seen as follows:

Occurrence	Level	Solution
none	0	(0,0,0)
$x_1 = 1$	1	(1,0,0)
$x_2 = 1$	1	(0,1,0)
$x_3 = 1$	1	(0,0,1)
$x_1 = 1, x_2 = 1$	2	(1,1,0)
$x_1 = 1, x_3 = 1$	2	(1,0,1)
$x_2 = 1, x_3 = 1$	2	(0,1,1)
$x_1 = 1, x_2 = 1, x_3 = 1$	3	(1,1,1)

The tree for this is

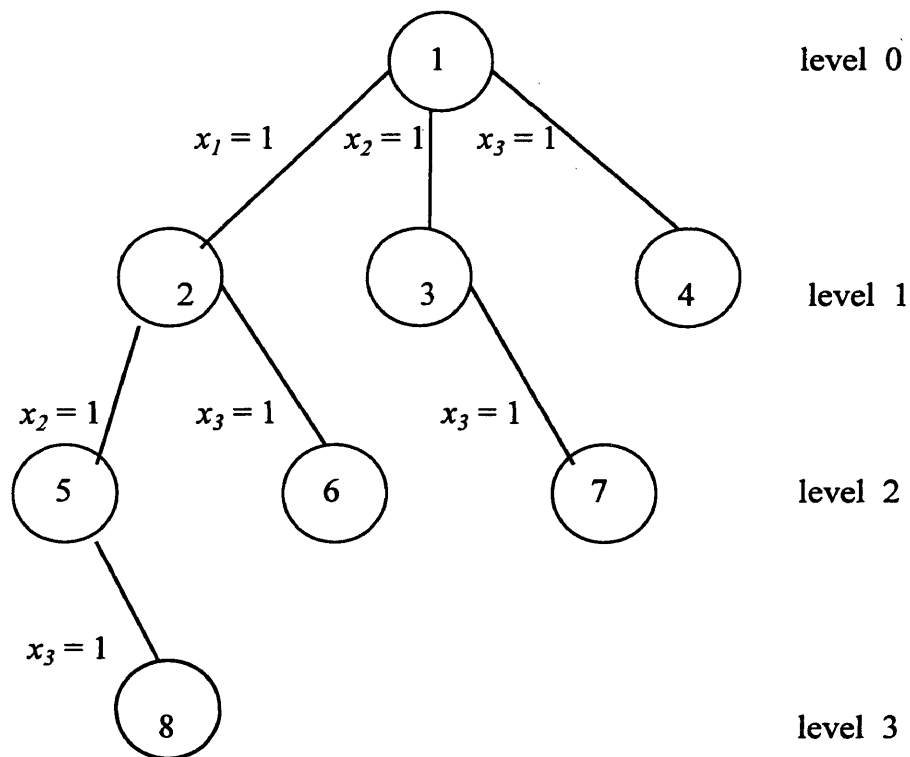


Figure 2.4 Branch and Bound Tree

Each branch can be seen as a subproblem; as the enumeration technique checks the branches, it eliminates the infeasible ones. To minimize the number of branches checked an algorithm can cut a branch if it becomes infeasible, therefore all solutions that are an extension of that branch are also cut. As the algorithm enumerates, it records the best feasible integer solution and as it comes across a better feasible solution, it replaces the one at hand. This process is continued until all branches are exhausted.

2.4.2 Balas Additive Algorithm:

This is the general form of a branch and bound algorithm. It was designed by Balas [18] and is made for 0-1 binary problems. The algorithm starts by setting all variables to zero and then systematically assigns variables the value of one in such a way that after trying a part of all the 2^n possible combinations, one either obtains an optimal solution or evidence that no feasible solution exists. The only operations required for this method are addition and subtraction, which is the reason it is called an “additive” algorithm.

The Balas algorithm requires the objective function to be in the form of minimization. A maximization problem can be reformulated by:

$$x_j = \begin{cases} x_j & \text{for } c_j \geq 0 \text{ when min; for } c_j \leq 0 \text{ when max} \\ 1 - x_j & \text{for } c_j < 0 \text{ when min; for } c_j > 0 \text{ when max} \end{cases}$$

where c_j is the coefficient of the objective function. The constraints must also be in the form of less than or equal to. “Ones complement” is another method of converting minimization problems to maximization problems and vice versa. An example using ones complement will be demonstrated in the test problems. Following the reformulation and the introduction of an m -component nonnegative slack vector \mathbf{y} (bold letters indicate vectors), the problem may be restated as:

$$\text{Minimize } z^* = \sum c_j x_j \quad (1)$$

$$\begin{aligned} \text{subject to} \quad & \sum a_{ij}x_j + y_i = b_i \quad (2) \quad \text{for } i = 1, 2, \dots, n \\ & x_j = 0 \text{ or } 1 \quad (3) \quad \text{for } j = 1, 2, \dots, m \\ & y_i \geq 0 \quad (4) \quad \text{(eq. 8)} \end{aligned}$$

where $c_j \geq 0$, and where x_j , c_j , a_{ij} , and b_i are obtained from x'_j , c'_j , a'_{ij} , and b'_i through the above described transformation. The dimension of x and c remains n . Furthermore, eq. 8 can be labeled as P.

A solution of P is designated as an $(n+m)$ dimensional vector $\mathbf{u}=(\mathbf{x},\mathbf{y})$ if it satisfies (2) and (3) of P; if it satisfies (2), (3), and (4) of P it is a feasible solution, and if it satisfies (1), (2), (3), and (4) it is an optimal solution. Now let P^s be the linear program defined by eq. 8 where we rewrite constraint (3) as

$$x_j \geq 0 \quad \text{for } j \in N \quad (3a)$$

$$x_j = 1 \quad \text{for } j \in J_s \quad (3b) \quad \text{(eq. 9)}$$

In this case J_s is a subset of N . If we let $J_0 = \phi$, then P^0 would be a new problem defined by (1), (2), (3a), and (4).

The idea behind the Balas algorithm follows this line of reasoning. Start with the regular linear program P^0 with $\mathbf{u}^0 = (\mathbf{x}^0, \mathbf{y}^0) = (\mathbf{0}, \mathbf{b})$, which is the dual-feasible solution to P^0 since $\mathbf{c} \geq \mathbf{0}$. The corresponding basis then consists of the unit-matrix $\mathbf{I}_m = \mathbf{e}_i$ for $i=1, 2, \dots, m$. Following this, a vector \mathbf{a}_{j_1} such that $a_{ij_1} < 0$ is chosen to enter the basis. The vector \mathbf{a}_{j_1} does not replace the vector \mathbf{e}_i as is the case in the dual simplex but the constraint $x_{j_1} = 1$ is added to P^0 to form $-x_{j_1} + y_{m+1} = -1$, where y_{m+1} is an artificial variable.

Upon completion we then have the problem P^1 with $J_1 = \{j_1\}$, which is eq. 8 and the additional constraint $x_{j_1} = 1$ (3b₁). Thus we now have the set $x_j = 0$ ($j \in N$), $y_i = b_i$ ($i \in M$), $y_{m+1} = -1$ which is the dual-feasible solution to P^1 . When considering the extended basis $\mathbf{I}_{m+1} = \mathbf{e}_i$ ($i = 1, 2, \dots, m+1$), one sees the \mathbf{e}_{m+1} vector corresponds to y_{m+1} . It is at this location (\mathbf{e}_{m+1}) that the vector \mathbf{a}_{j_1} is introduced and hence x_{j_1} assumes the value of 1 in the dual feasible solution P^1 . This solution can then be written as $\mathbf{u}^1 = (\mathbf{x}^1, \mathbf{y}^1) = (x_{j_1}^1, \dots, x_m^1, y_{j_1}^1, \dots, y_m^1)$ because the artificial variable y_{m+1} becomes 0. Consequently, the new dual-feasible solution $\mathbf{u}^1 = (\mathbf{x}^1, \mathbf{y}^1)$ to P^1 is

$$x_j^1 = \begin{cases} 1 & (j = j_1) \\ 0 & (j = N - \{j_1\}) \end{cases} \quad y_i^1 = b_i - a_{ij_1} \quad \text{for } i \in M \quad (\text{eq. 10})$$

If the solution vector \mathbf{u}^1 continues to have negative components, then another vector \mathbf{a}_{j_2} must be selected to enter the basis, and a new constraint $x_{j_2} = 1$ must be added to P^1 in the same form as above $-x_{j_2} + y_{m+2} = -1$, where y_{m+2} is an artificial variable. We now have the problem P^2 again consisting of (eq. 8) and the additional constraint set (3b₂) which is $x_{j_1} = 1$ and $x_{j_2} = 1$. The dual-feasible solution to P^2 is therefore the set $x_{j_1} = 1$, $x_j = 0$ [$j \in (N - \{j_1\})$], $y_i = b_i - a_{ij_1}$ ($i \in M$), $y_{m+2} = -1$. Repeating the same procedure as previously described, the vector \mathbf{a}_{j_2} replaces \mathbf{e}_{m+2} and x_{j_2} assumes the value of 1. The new dual feasible solution to P^2 is $\mathbf{u}^2 = (\mathbf{x}^2, \mathbf{y}^2)$, where

$$x_j^2 = \begin{cases} 1 & (j = j_1, j_2) \\ 0 & (j = N - \{j_1, j_2\}) \end{cases} \quad y_i^2 = y_i^1 - a_{ij_2} \quad \text{for } i \in M$$

This procedure is continued until a solution is reached with all components nonnegative or until the algorithm indicates that no feasible solution exists. During the process, a sub-optimal solution may be found when a nonnegative vector $\mathbf{u}^s = (\mathbf{x}^s, \mathbf{y}^s)$ is obtained. This solution may not be the optimal solution for P but it will be a feasible solution to P.

This procedure of the algorithm is then repeated again, starting from a solution \mathbf{u}^p ($p < s$) until another feasible solution \mathbf{u}^t is found such that $z_t < z_s$ or evidence indicates that no other feasible solution exists. In this case the algorithm terminates.

The Balas algorithm essentially follows only some of the branches of the tree, while disregarding most of the others. By setting all of the variables equal to 0 and then systematically assigning 1 to some of the variables, you can minimize the number of 2^n possible combinations needed to check before obtaining either an optimal solution or evidence that no feasible solution exists. This is accomplished by (a) carefully considering at each iteration a subset of variables that are candidates for assuming the value of 1 and (b) selecting the best variable from among the candidates [18].

This is just one example of a branch and bound algorithm. There are numerous types that have been developed in the last thirty years. This particular one is specifically designed for a binary integer problem; this algorithm would not work for an all integer problem. In that case another algorithm would have to be considered. The fundamental principles are the same for branch and bound problems; that is, systematically searching each lattice point in the solution space for the optimal solution.

Chapter 3

MULTIPLE CHOICE PROGRAMMING

3.1 General Discussion:

Multiple Choice Programming is a procedure developed for linear programming with variables that must assume zero-one as the solution. MCP can also be used in problems that have continuous variables as long as 0-1 variables are involved. The 0-1 variables are constrained such that $\sum x_j = 1$, $\sum y_j = 1$, $\sum z_j = 1$, etc. The idea is to find the optimum choice for the 0-1 variables that will assume the value of one, and allow the continuous variables to go to their optimal solution so as to maximize the objective function. This is done through the successive modifications of a linear programming problem that converges to the optimal solution. The algorithm has the following characteristics:

1. One additional profit constraint is required for each set of 0-1 integers that sum to one.
2. These extra profit constraints are formulated such that they explicitly limit the objective function by using profit bounds described later in this chapter.

3. The procedure of MCP is a cycle that continues to tighten the extra profit constraints and eventually solves a linear programming problem equivalent to the desired integer programming problem.

4. The method uses profit gradients calculated from noninteger solutions to force the variables to 0-1 [10].

Thus the MCP problems are seen as linear programming problems that have a single objective function and have constraints that include integer sum (0-1) constraints. For each integer sum constraint, we add a profit constraint. For example, if a problem has six integer sum constraints, there will be six additional profit constraints. It is easy to see that the problem size can grow very quickly. This is especially true for capital budgeting problems because this specific class of problems usually does not have integer sum constraints. Therefore, such constraints must be added to the problem before using MCP.

The following sections will describe the algorithm in detail. There will also be a section describing the reformulation of a typical capital budgeting problem so that it can be solved using MCP. The final portion of this chapter will give two example problems worked out in detail.

3.2 Description of the Algorithm:

The theory behind MCP is that there are inequalities on the objective function after running an LP iteration, meaning the solutions do not usually have integer values.

However, the variables can then be forced to an integer value (0-1 because of the integer sum constraints), so that the least amount of cost to the objective function will be incurred. By continually tightening the profit constraints, the objective function will decrease by a fraction of the shadow price and the variables will converge to 0-1 [10].

Capital budgeting problems are ideal for MCP because, with the exception of the integer sum constraint, they are both in the same form:

$$\begin{aligned}
 &\text{Maximize} && z^* = \sum_{j=1}^n c_j x_j \\
 &\text{subject to:} && \sum_{j=1}^n a_{ij} x_j \leq b_i \quad \text{for } i=1,2,\dots,m; \quad a_{ij} \geq 0 \quad \forall i, \forall j \\
 &&& x_j \geq 0 \quad \text{for } j=1,2,\dots,n. \quad \text{(eq. 1)}
 \end{aligned}$$

The additional integer sum constraints, however, can be added to the problem so that the capital budgeting problem now becomes an MCP problem. Before continuing, I will introduce some additional terminology and notation that will be used throughout this chapter. As aforementioned, a set of variables x_1, x_2, \dots, x_n which sum to one is referred to as an *integer set*. The constraint $\sum x_n = 1$ which this leads to is an *integer sum*

constraint. For each integer sum constraint a problem possesses, an additional row will be added to the problem; this row is known as a *profit constraint*. As MCP solves the problem, it cycles through a series of LP solutions; each of these completed LP solutions is referred to as a *trial*. The trials will be denoted by t . I will use the standard notation in LP for the objective function as z^* but the value of the objective function solution at the end of trial t will be identified by $\phi^{(t)}$. Additional notation includes $x_j^{(t)}$ which represents the value of variable x_j at trial t and $r_j^{(t)}$ which is called the *profit gradient* for x_j computed at trial t . The profit gradient is explained in detail later in this section. At the conclusion of each trial, a *profit bound* $\phi_j^{(t)}$ is computed for each variable in the objective function such that $z^* \leq \phi_j^{(t)}$ if $x_j = 1$. Thus, the profit bound is developed by $\phi_j^{(t)} = \phi^{(t)} - (1 - x_j^{(t)}) r_j^{(t)}$. This leads to the most important part of MCP, the profit constraint at trial T . This additional constraint says that the objective function at the MCP optimum cannot exceed the weighted average minimum profit bound and is denoted by:

$$z^* \leq \sum_{j=1}^p x_j^* \min_{t \leq T} \phi_j^{(t)} \quad [10] \quad (\text{eq. 11})$$

The next portion presented here deals with the calculation of the profit gradient $r_j^{(t)}$, this is closely related to $-\partial z^* / \partial x_j$ calculated at trial t . Additionally, *exclusion sets* will be introduced. The purpose of exclusion sets is to minimize the effect of zero reduced costs (shadow prices). This is the cost incurred on the objective function if a variable not in the

basis is forced into the basis. Exclusion sets S_j are defined as $S_j = \{X_k: x_j=1 \Rightarrow X_k=0\}, j = 1, 2, \dots, p$; that is, when $x_j=1$ the other variables in the exclusion set must be equal to zero. Elements in the final tableau of the LP are denoted by $a_{ij}^{(t)}$ where i is the row and j is the column at trial t . The reduced cost row will be denoted as $a_{0k}^{(t)}$, which identifies the reduced cost in row zero variable k at trial t . The profit gradient is then defined as:

$$r_j^{(t)} = \begin{cases} a_{0k}^{(t)} & \text{if } x_j \text{ is non basic and occupies column } k \\ \min_{a_{ik}^{(t)} < 0} \frac{a_{0k}^{(t)}}{a_{ik}^{(t)}} & \text{if } x_j \text{ is basic and occupies row } i, \text{ take the minimum ratio} \\ & \text{using the negative entries of the row and deleting all } S_j \text{ columns} \end{cases}$$

(eq. 12)

Upon calculating the profit gradient, the profit bound must be calculated. This in turn is used to calculate the profit constraint. The profit bound defined earlier, $\phi_j^{(t)} = z^{(t)} - (1 - x_j^{(t)}) r_j^{(t)}$, limits the upper bound of the objective function such that when the profit constraint is calculated using eq. 11, the profit bound for variable j at trial t will always be less than or equal to its previous value [12]. In order to properly form the profit constraint, some simple algebra must be done. Notice that eq. 11 is not in the form of a constraint for LP. After substituting values into eq. 11, bring all of the variables to the left side of the inequality such that the constraint is always less than or equal to zero. This cycle is repeated for each LP trial, until the problem converges to a 0-1 solution.

The final part of the algorithm is the stopping criterion. The MCP procedure has converged to a solution when all of the variables required to be integer fall within the convergence criterion. This convergence criterion is defined as $x_j \geq \varepsilon$ or $x_j \leq 1 - \varepsilon$ for all j in the same trial. In other words, all of the variables must meet this criterion in the same trial otherwise the algorithm continues. When the variables meet this criterion, they are forced to their respective values: zero or one.

3.3 Reformulation of Capital Budgeting Problems:

Several times, it has been mentioned that capital budgeting problems as they stand do not meet the requirement to be solved by MCP. They usually do not have integer sum constraints as part of their formulation and are found in the form of eq. 1. After reformulating these problems they will appear as:

$$\text{Maximize } z^* = \sum_{j=1}^n c_j x_j$$

$$\text{subject to: } \sum_{j=1}^n a_{ij} x_j \leq b_i \quad \text{for } i=1,2,\dots,m; \quad a_{ij} \geq 0 \quad \forall i, \forall j$$

$$\sum_{j=1}^n a_{ij} x_j = 1 \quad \text{for } i=m+1,\dots,r; \quad \text{and } a_{ij}=0 \text{ or } 1 \quad \forall i, \forall j$$

$$x_j \geq 0 \quad \text{for } j=1,2,\dots,n. \quad (\text{eq. 13})$$

Therefore added steps are implemented prior to the first LP trial. These steps are as follows:

1. Add one “dummy” variable with the coefficient of zero in the objective function for each original variable. If the original problem has two such original variables, the reformulated problem will have four variables.

2. Add one integer sum constraint for each non-zero variable such that a “1” in the constraint corresponds to a non-zero coefficient in the objective function and another “1” in the same constraint corresponds to a zero coefficient in the objective function.

For example, suppose we consider a fictitious company XYZ Corporation. XYZ Corporation is considering which projects to undertake in the next fiscal year to maximize its return on investment. The net present value in current dollars for project 1 is c_1 and for project 2 is c_2 . There are two limiting factors that the company must consider: (1) the budget, which is b_1 and (2) the man hours allotted in the next fiscal year, which is b_2 . In order to do project 1, a_{11} dollars are needed and a_{21} man hours are required. Project 2 requires a_{21} dollars and a_{22} man hours. This problem would appear mathematically as:

$$\begin{aligned} \text{Maximize } z^* &= c_1x_1 + c_2x_2 \\ \text{subject to } & a_{11}x_1 + a_{12}x_2 \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 \leq b_2 \end{aligned}$$

After reformulating the problem it would appear as:

$$\begin{aligned} \text{Maximize } z^* &= c_1x_1 + c_2x_2 + 0x_3 + 0x_4 \\ \text{subject to } & a_{11}x_1 + a_{12}x_2 \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 \leq b_2 \\ & x_1 + x_3 = 1 \\ & x_2 + x_4 = 1. \end{aligned}$$

These integer sum constraints are forcing the variables to be values between 0-1 and the second variable in each integer sum constraint represents a zero in the objective function. It is clear to see, however, that these problems can grow quite large; a problem with 10 variables and 10 constraints after reformulation will have 20 variables and 20 constraints. This does not include the profit constraints which will add another 10 constraints to this problem. It is important here to realize that we have increased the size of a linear programming problem, not an integer programming problem. This is true because we are solving a sequence of linear programming problems rather than integer programming problems.

3.4 Detailed Description:

This section will explain the algorithm and will include a step-by-step process to follow. There are two key areas that make this algorithm unique from other cutting plane methods, the calculation of the profit gradient and the calculation of the additional profit constraints. The profit gradient leads to the profit constraint which is the motivating force behind driving the solution to 0-1.

The profit gradient is calculated from the final tableau of the LP simplex method. This is done by considering all of the variables in the basis; if a variable in the basis is a slack variable we disregard it because it cannot provide any important information to us. We consider only variables x_j . Recall that the profit gradient is defined as:

$$r_j^{(t)} = \begin{cases} a_{ok}^{(t)} & \text{if } x_j \text{ is non basic and occupies column } k \\ \min_{a_{ik}^{(t)} < 0} \frac{a_{ok}^{(t)}}{a_{ik}^{(t)}} & \text{if } x_j \text{ is basic and occupies row } i, \text{ take the minimum ratio} \\ & \text{using the negative entries of the row and deleting all } S_j \text{ columns} \end{cases}$$

This basically tells the user that the profit gradient is equal to the reduced cost of a variable if it is nonbasic; otherwise it is equal to the minimum ratio of the negative entries of the variable row to the respective reduced cost. The user must identify the variable row, scan across the row until a negative entry is found, and divide that entry by its

corresponding reduced cost. The procedure is continued until all entries in that row have been checked; the profit gradient is then equal to the minimum of these values considered. For example, suppose the final tableau is as follows:

$$\begin{bmatrix} c_1 & c_2 & \cdot & \cdot & c_n & b_i \\ a_{11} & a_{12} & \cdot & \cdot & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdot & \cdot & a_{2n} & b_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ .203 & -.62 & .1 & -.23 & .4 & -.52 & .34 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \cdot & \cdot & \cdot & a_{nn} & b_n \\ * & -1.426 & * & -.782 & * & -.936 & 10 \end{bmatrix}$$

Figure 3.1 LP Final Tableau

Furthermore, suppose that the first row of numbers is row i and the last row is the reduced cost row with 10 being the solution to the objective function in this trial. The profit gradient for row i would then be the minimum of $\{-1.426/-.62=2.3; -.782/-.23=3.4; -.936/-.52=1.8\}$, thus $r_i^{(i)}=1.8$. If there is no negative value in the row being considered, the profit gradient is equal to the value of big "M". Big "M" is an extremely large value and says the value of x_j cannot be increased, so it is as large as it can possibly become. It

is important to note that the profit gradient is *always* a positive value. If the profit gradient were negative, the algorithm would not converge and the solution space would either become increasingly large or remain the same.

The calculation of the profit bound then follows the profit gradient. As defined, the profit bound is $\phi_j^{(t)} = z^{(t)} - (1 - x_j^{(t)}) r_j^{(t)}$ where $z^{(t)}$ is the value of the objective function at trial t and $x_j^{(t)}$ is the value of variable x_j at trial t . Each variable in the integer set will have a profit bound associated with it. The profit bound similar to the profit gradient will *always* be positive. The calculation for this is simple substitution; refer back to the final tableau above. In this tableau, $z^{(t)}=10$, $x_i^{(t)}=.34$, and $r_i^{(t)}=1.8$, hence the profit bound would be $\phi_j^{(t)} = 10 - (1 - .34) * 1.8$ or $\phi_j^{(t)}=8.812$.

The final and most important part of MCP is the actual construction of the profit constraint. For each integer sum constraint, there will be a corresponding profit. Recall that the profit constraint is defined as:

$$z^* \leq \sum_{j=1}^p x_j * \min_{i \leq T} \phi_j^{(i)} \quad (\text{eq. 11})$$

where z^* is the objective function. This says that the objective function is less than or equal to the sum of the integer sum values multiplied by their minimum profit bound. It is important to note that the equation refers to the minimum profit bound for variable x_j over all the trials that have been completed. In the example above, $\phi_j^{(t)}=8.812$ in trial t ; if

in the previous trial $t-1$ the profit bound was $\phi_j^{(t-1)}=7.5$ then the profit constraint would use $\phi_j^{(t-1)}=7.5$. Lastly, the variables on the right side of eq. 11 must be brought over to the left side. This is done by subtracting the right side values from both sides and combining the like terms. In our example above suppose that $n=4$ so there are four variables in the problem. Furthermore, suppose $c_1 = 4$, $c_2 = 5$, $c_3 = 0$, $c_4 = 0$, $\phi_1^{(2)} = 7.5$, and $\phi_3^{(2)} = 8$. The profit constraint using equation 11 would then be set up as:

$$4x_1 + 5x_2 + 0x_3 + 0x_4 \leq 7.5x_1 + 0x_2 + 8x_3 + 0x_4$$

After simple algebra, the profit constraint that would go into the LP would be:

$$-3.5x_1 + 5x_2 - 8x_3 \leq 0.$$

Notice that a profit bound is only put into eq. 11 for corresponding integer sum values. In this case there were only two non-zero variables in the objective function, so variables x_1 and x_3 form one integer sum constraint. Corresponding to the x_1 and x_3 integer sum constraint is a profit constraint.

In order to calculate the additional constraints, one must use the final tableau from the simplex method. This implies that the initial LP will have the integer sum constraints but not the profit constraints. Therefore the initial trial in the MCP algorithm can be seen as trial 0. The actual MCP procedure does not begin until the first LP is complete and

convergence of the variables is checked. If all of the variables are 0-1, then the algorithm terminates and MCP is not used in the problem. This occurs very rarely, however.

3.4.1 Algorithm to Solve Capital Budgeting Problems:

This section details the algorithm in a step by step manner. It also shows a flowchart for the algorithm to allow the user to follow the steps.

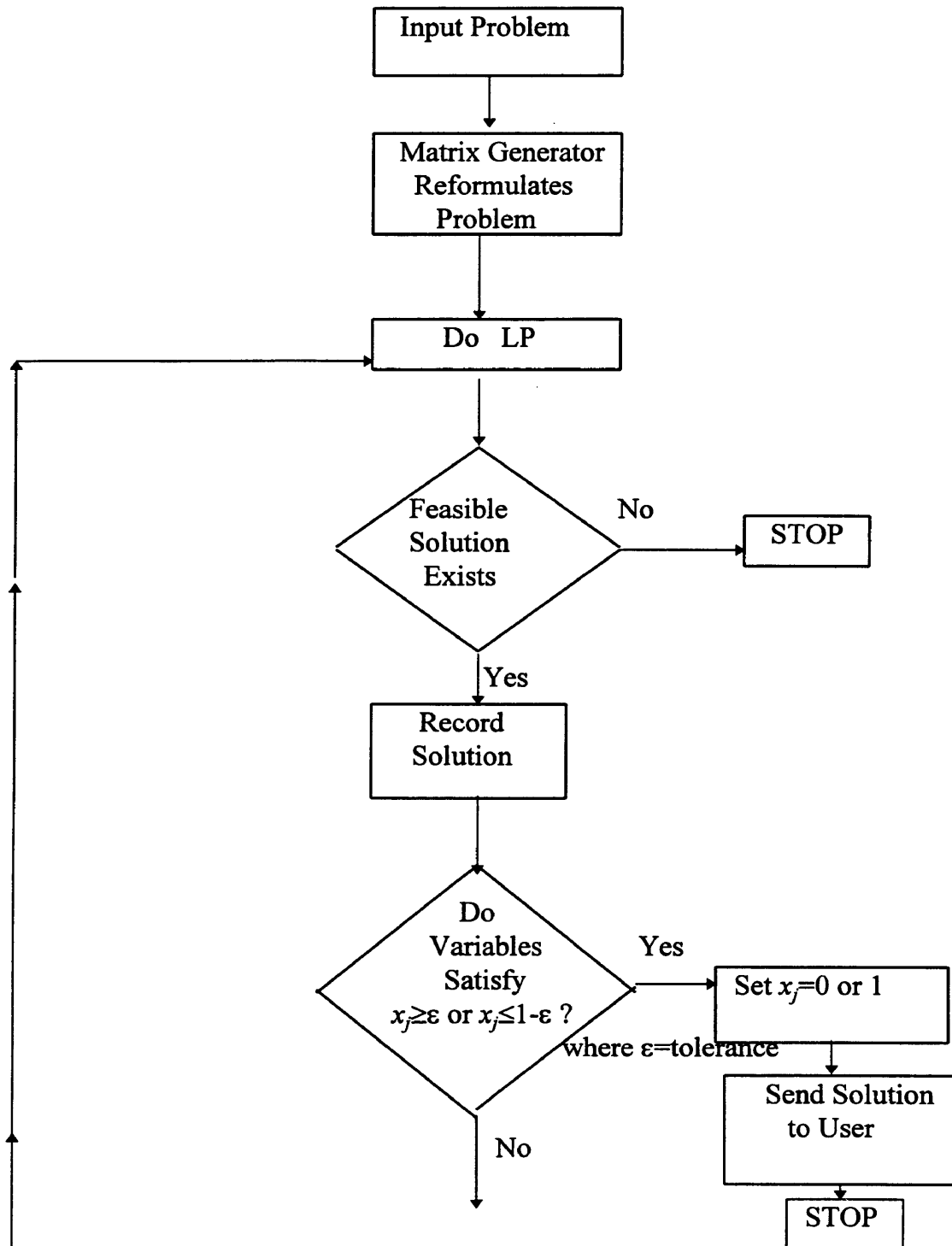
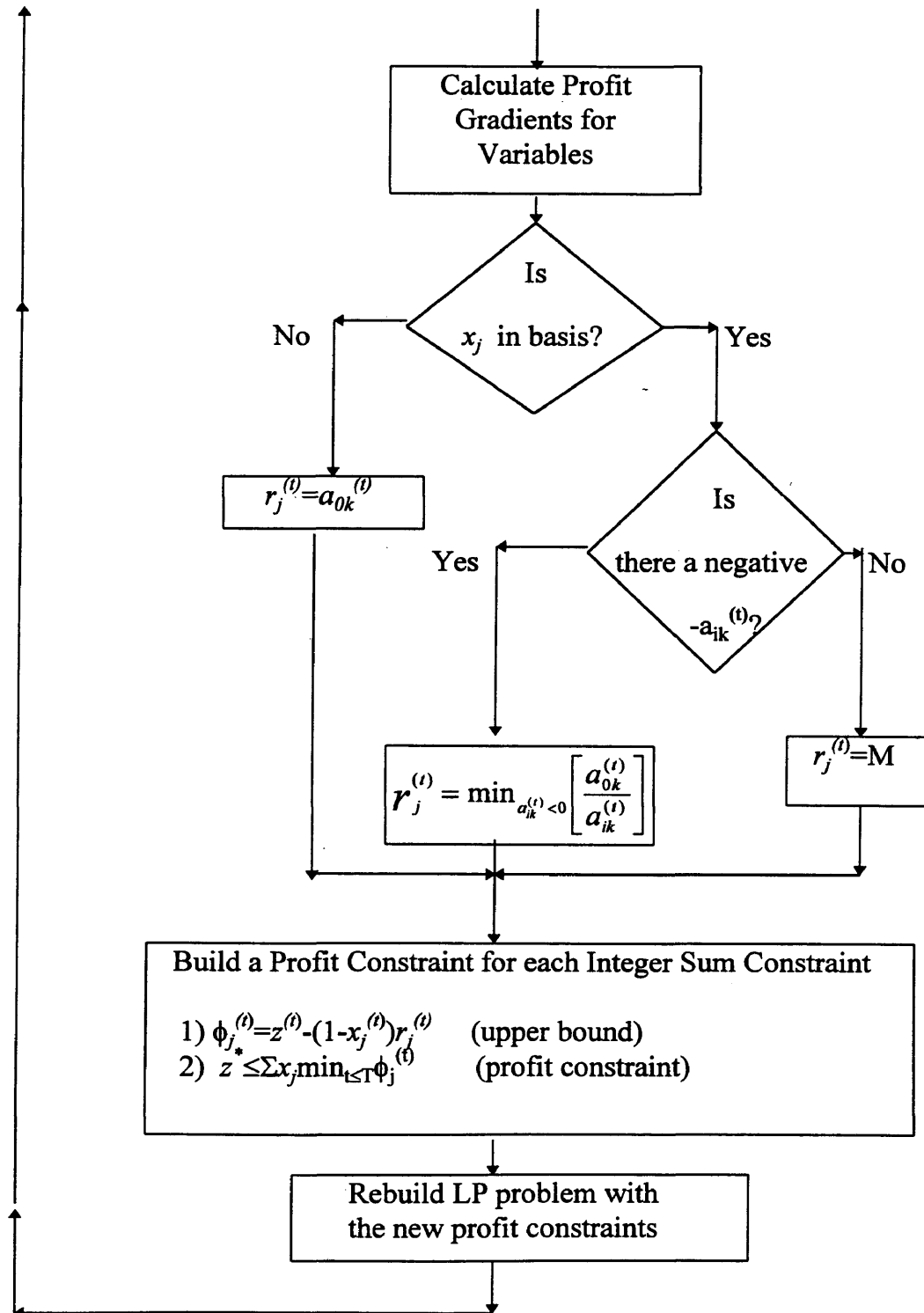


Figure 3.2 Algorithm Flow Chart

Figure 3.2 Continued



The following steps detail the algorithm. The code that accompanies the algorithm is located in appendix B.

Step 1: Input the problem.

Step 2: For each variable add one “dummy” variable with a coefficient of zero to the objective function. The problem should now have two times the number of variables as the original input.

Step 3: Add one integer sum constraint for each nonzero variable in the objective function. The first variable and the first “dummy” variable will compose the first integer sum constraint. The second variable and the second “dummy” variable will make up the second constraint, etc. In each case they are equal to one.

Step 4: Run problem as an LP.

If feasible solution does not exist, go to step 13.

Step 5: Record the feasible solution.

Step 6: If all variables satisfy $x_j \geq \varepsilon$ or $x_j \leq 1 - \varepsilon$ (where ε is the tolerance defined by the user), go to step 11.

Step 7: Calculate the profit gradients for each variable.

- a. Is x_j in the basis? If it is not in the basis, then go to step 7e.
- b. Scan across the row for a negative a_{ij} , divide each negative a_{ij} into its respective reduced cost (a_{0k}). If there are no negative values, then go to step 7d.
- c. Set $r_j^{(t)} = \min_{a_{ik}^{(t)} < 0} \frac{a_{0k}^{(t)}}{a_{ik}^{(t)}}$, then go to step 7f.
- d. Set $r_j^{(t)} = M$, then go to step 7f.
- e. Set $r_j^{(t)} = a_{0k}^{(t)}$ (reduced cost), then go to step 7f.
- f. Find the next x_j and go to step 7a. When all x_j variables are have been evaluated, go to step 8.

Step 8. Calculate the profit bounds for each variable.

- a. All profit bounds follow: $\phi_j^{(t)} = z^{(t)} - (1 - x_j^{(t)})r_j^{(t)}$.

Step 9. Build profit constraints; for each integer sum constraint there will be one profit constraint.

- a. All profit constraints follow: $z^* \leq \sum x_j^* \min_{t \leq T} \phi_j^{(t)}$.
- b. Bring all variables on the right hand side of the equation in step 9a to the left side by subtracting. The constraint should be in the form of less than or equal to zero.

Step 10. Rebuild the LP problem so it is identical to the first LP run, then add the additional profit constraints. Go to step 4.

Step 11. Force variables to their respective value (0 or 1).

Step 12. Replace variables with their solutions in the objective function to obtain the optimal value of z^* . Send solution to user.

Step 13. Stop.

3.5 Example Problems:

This section gives two example problems. The first problem is very small and can be solved by inspection but, the second problem is more difficult. In each case, the example includes two iterations of the MCP algorithm and then sums up the solution for each subsequent trial. Both of these example problems require numerous trials before they converge. Since the algorithm continues in a repeating manner, it is not necessary to continue the cyclic process once the reader grasps the concept.

3.5.1 Example Problem 1:

Consider the capital budgeting problem:

$$\begin{aligned}
 \text{Max } z^* &= 6x_1 + 3x_2 + 4x_3 + 8x_4 \\
 \text{st} \quad & -2x_1 + 4x_3 + 2x_4 \leq 5 \\
 & 3x_1 + x_2 + 2x_3 + 2x_4 \leq 4 \\
 & x_1 + x_2 + x_3 + x_4 \leq 2 \\
 & x_i \geq 0 \quad \text{for } i = 1, 2, 3, 4
 \end{aligned}$$

This problem can be solved using MCP as follows:

Step 1. Input the problem.

Step 2 and 3. Add one additional variable for each original variable. Additionally, add one integer sum constraint for each nonzero variable. Thus, the problem to be solved now appears as:

$$\begin{aligned}
 \text{Max } z^* &= 6x_1 + 3x_2 + 4x_3 + 8x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 \\
 \text{st} \quad & -2x_1 + 4x_3 + 2x_4 \leq 5 \\
 & 3x_1 + x_2 + 2x_3 + 2x_4 \leq 4 \\
 & x_1 + x_2 + x_3 + x_4 \leq 2 \\
 & x_1 + x_5 = 1 \\
 & x_2 + x_6 = 1
 \end{aligned}$$

$$\begin{aligned}
 x_3 + x_7 &= 1 \\
 x_4 + x_8 &= 1 \\
 x_i &\geq 0 \quad \text{for } i = 1, 2, 3, 4, 5, 6, 7, 8
 \end{aligned}$$

Step 4. Run the problem from step 3 through an LP solver.

Final Tableau after initial LP before 1st MCP

Basis	x_3	x_8	S2	S3	RHS
S1	5	-3	1	-1	4
x_1	0.5	-0.5	0.5	-0.5	0.5
x_2	0.5	-0.5	-0.5	1.5	0.5
x_5	-0.5	0.5	-0.5	0.5	0.5
x_6	-0.5	0.5	0.5	-1.5	0.5
x_7	1	0	0	0	1
x_4	0	1	0	0	1
Red. Cost	-0.5	-3.5	-1.5	-1.5	12.5

Figure 3.3 Example 1 Tableau MCP Trial (0)

Step 5. Record the feasible solution. The variables in the basis are those variables listed in the column designated “basis”. The basis variables in this trial are $x_1^{(0)} = .5$, $x_2^{(0)} = .5$, $x_4^{(0)} = 1$, $x_5^{(0)} = .5$, $x_6^{(0)} = .5$, and $x_7^{(0)} = 1$. The term “S1” is a slack variable so it does not impact on the MCP. The non-basis variables are x_3 and x_8 and the value of both of these variables is zero. Finally, the optimal objective function value in this trial is $z^{(0)} = 12.5$.

Step 6. Are all of the variables within a tolerance of .95 (the user designated tolerance)?

This means all of the variables must be greater than or equal to .95 or less than or equal to .05. Since this is not the case, proceed to step 7.

Step 7. Calculate the profit gradients.

a. The basis variables are $x_1, x_2, x_4, x_5, x_6,$ and x_7 . The non-basis variable are x_3 and x_8 .

b. The profit gradients are then

$$r_1^{(0)} = \min \{-3.5/-0.5, -1.5/0.5\} = 3$$

$$r_2^{(0)} = \min \{-3.5/-0.5, -1.5/-0.5\} = 3$$

$$r_3^{(0)} = .5$$

$$r_4^{(0)} = M$$

$$r_5^{(0)} = \min\{-0.5/-0.5, -1.5/-0.5\} = 1$$

$$r_6^{(0)} = \min \{-0.5/-0.5, -1.5/-0.5\} = 1$$

$$r_7^{(0)} = M$$

$$r_8^{(0)} = 3.5$$

Step 8: Calculate the profit bounds

a. $\phi_1^{(0)} = 12.5 - (1 - 0.5)3 = 11$

$$\phi_2^{(0)} = 12.5 - (1 - 0.5)3 = 11$$

$$\phi_3^{(0)} = 12.5 - 0.5 = 12$$

$$\phi_4^{(0)} = 12.5 - (1 - 1)M = 12.5$$

$$\phi_5^{(0)} = 12.5 - (1 - 0.5)1 = 12$$

$$\phi_6^{(0)} = 12.5 - (1 - 0.5)1 = 12$$

$$\phi_1^{(0)} = 12.5 - (1-1)M = 12.5$$

$$\phi_1^{(1)} = 12.5 - 3.5 = 9$$

Step 9: Build the profit constraints

a. $6x_1 + 3x_2 + 4x_3 + 8x_4 \leq 11x_1 + 12x_5$

$$6x_1 + 3x_2 + 4x_3 + 8x_4 \leq 11x_2 + 12x_6$$

$$6x_1 + 3x_2 + 4x_3 + 8x_4 \leq 12x_3 + 12.5x_7$$

$$6x_1 + 3x_2 + 4x_3 + 8x_4 \leq 12.5x_4 + 9x_8$$

b. These profit constraints are then rewritten as:

$$-5x_1 + 3x_2 + 4x_3 + 8x_4 - 12x_5 \leq 0$$

$$6x_1 - 8x_2 + 4x_3 + 8x_4 - 12x_6 \leq 0$$

$$6x_1 + 3x_2 - 8x_3 + 8x_4 - 12.5x_7 \leq 0$$

$$6x_1 + 3x_2 + 4x_3 - 4.5x_4 - 9x_8 \leq 0$$

Step 10: Rebuild the LP problem and return to step 4. The new LP problem will be the same as the original plus the profit constraints; in other words, the problem will now have eleven constraints.

Step 4: Run the problem through the LP again.

Final Tableau after 1st MCP

Basis	x_8	S1	S8	S9	RHS
S11	1.5	-0.0417	-0.333	-0.125	0.5833
S2	1.5	-0.0833	-0.667	0.25	0.0833
S3	0.75	-0.0417	0.167	-0.375	0.0417
x_2	-0.5	-0.0833	-0.667	0.75	0.0833
x_1	-0.5	-0.0833	0.333	-0.25	0.0833
S10	-0.015	-0.0234	-0.094	-0.023	0.1875
x_4	1	0	0	0	1
x_5	0.5	0.0833	-0.333	0.25	0.9167
x_6	0.5	0.0833	0.667	-0.75	0.9167
x_7	0.75	-0.2083	-0.167	0.125	0.2083
x_3	-0.75	0.2083	0.167	-0.125	0.7917
Red. Cost	-0.5	-0.0833	-0.667	-0.25	11.917

Figure 3.4 Example 1 Tableau MCP Trial (1)

Step 5. Record feasible solution. The solution for this iteration is $x_1^{(1)} = .0833$,

$x_2^{(1)} = .0833$, $x_3^{(1)} = .7917$, $x_4^{(1)} = 1$, $x_5^{(1)} = .9167$, $x_6^{(1)} = .9167$, and $x_7^{(1)} = .2083$. The optimal objective function value is $z^{(1)} = 11.917$.

Step 6. Check the tolerance level of each variable. Since the variables are not within the tolerance, continue to step 7.

Step 7. Calculate profit gradients.

$$r_1^{(1)} = \min\{-.5/-.5, -.083/-.083, -.25/-.25\} = 1$$

$$r_2^{(1)} = \min\{-.5/-.5, -.083/-.083, -.667/-.667\} = 1$$

$$r_3^{(1)} = \min\{-.5/-.75, -.25/-.125\} = .667$$

$$r_4^{(1)} = M$$

$$r_5^{(1)} = -.667/-.333 = 2$$

$$r_6^{(1)} = -.25/-.75 = .333$$

$$r_7^{(1)} = \min \{-.083/-.2083, .667/-.167\} = .4$$

$$r_8^{(1)} = .5$$

Step 8. Calculate the profit bounds.

$$\phi_1^{(1)} = 11.92 - (1 - .0833)1 = 11$$

$$\phi_2^{(1)} = 11.92 - (1 - .0833)1 = 11$$

$$\phi_3^{(1)} = 11.92 - (1 - .79).667 = 11.78$$

$$\phi_4^{(1)} = 11.92 - (1 - 1)M = 11.92$$

$$\phi_5^{(1)} = 11.92 - (1 - .92)2 = 11.76$$

$$\phi_6^{(1)} = 11.92 - (1 - .92).333 = 11.89$$

$$\phi_7^{(1)} = 11.92 - (1 - .21).4 = 11.6$$

$$\phi_8^{(1)} = 11.92 - .5 = 11.42$$

Step 9. Build the profit constraints.

$$-5x_1 + 3x_2 + 4x_3 + 8x_4 - 11.76x_5 \leq 0$$

$$6x_1 - 8x_2 + 4x_3 + 8x_4 - 11.89x_6 \leq 0$$

$$6x_1 + 3x_2 - 7.78x_3 + 8x_4 - 11.6x_7 \leq 0$$

$$6x_1 + 3x_2 + 4x_3 - 3.92x_4 - 9x_8 \leq 0$$

Step 10. Rebuild the LP replacing the previous profit constraints with the profit constraints derived from trial 1. Go back to step 4.

This process is continued until the variables are within the tolerance set by the user.

When this condition is met, the variables are forced to either zero or one. The following table depicts the solutions at each iteration:

MCP TRIAL NUMBER	VARIABLES	SOLUTION
2	$x_1=.0249, x_2=.1773, x_3=.7625, x_4=1, x_5=.9751, x_6=.8227,$ $x_7=.2375$	$z = 11.7313$
3	$x_1=.113, x_3=.8329, x_4=.9472, x_5=.887, x_6=1, x_7=.1671,$ $x_8=.0528$	$z = 11.5871$
4	$x_1=.3077, x_2=.3157, x_3=.1522, x_4=.9999, x_5=.6923, x_6=.6843,$ $x_7=.8478$	$z = 11.4018$
5	$x_1=.1436, x_2=.355, x_3=.3249, x_4=1, x_5=.8564, x_6=.645,$ $x_7=.6751$	$z = 11.2264$
6	$x_1=.2429, x_2=.2948, x_3=.1988, x_4=1, x_5=.7571, x_6=.7052,$ $x_7=.8011$	$z = 11.137$
7	$x_1=.1707, x_2=.3589, x_3=.244, x_4=1, x_5=.8293, x_6=.6411,$ $x_7=.756$	$z = 11.0766$
8	$x_1=.2217, x_2=.3311, x_3=.1780, x_4=1, x_5=.7784, x_6=.6689,$ $x_7=.82$	$z = 11.043$
9	$x_1=.186, x_2=.4062, x_3=.1717, x_4=1, x_5=.814, x_6=.5938,$ $x_7=.8283$	$z = 11.0214$
10	$x_1=.2133, x_2=.4202, x_3=.1173, x_4=1, x_5=.7867, x_6=.5798,$ $x_7=.8827$	$z = 11.0098$
11	$x_1=.1789, x_2=.5364, x_3=.0803, x_4=.9999, x_5=.8211, x_6=.4636,$ $x_7=.9198$	$z = 11.0034$
12	$x_1=.1723, x_2=.6074, x_3=.0363, x_4=1, x_5=.8277, x_6=.3926,$ $x_7=.9637$	$z = 11.0009$
13	$x_1=.1075, x_2=.7686, x_3=.0124, x_4=1, x_5=.8926, x_6=.2314,$ $x_7=.9876$	$z = 11.0001$

Figure 3.5 Solutions for Example 1

After step 4 of MCP trial 14, we go to step 5.

Step 5. Record the feasible solution.

$$x_1^{(14)} = 0$$

$$x_2^{(14)} = .999997$$

$$x_3^{(14)} = 0$$

$$x_4^{(14)} = .999998$$

$$x_5^{(14)} = 1$$

$$x_6^{(14)} = .000003$$

$$x_7^{(14)} = 1$$

$$x_8^{(14)} = .000002$$

$$z^{(14)} = 11.00009$$

Step 6. Do all variables satisfy the tolerance? Yes, they do; then go to step 11. If not, then continue on to step 7.

Step 11. Force the variables to their respective values. The solution now become $x_1=0$, $x_2=1$, $x_3=0$, and $x_4=1$. The optimal objective function value is therefore $z^* = 11$. Stop.

This example problem took fourteen MCP trials to find the solution and completed it in approximately 15 seconds (486DX, 33mhz). The tolerance in this example problem did not play as important a role as is sometimes possible. In this case, the tolerance could have been set at .9 and it would have still taken 14 trials to reach the solution.

3.5.2 Example Problem 2:

It was not difficult to see what the solution was in the first example problem by simply looking at it for a few minutes. This problem is larger, so it is not as easy to look at it and find the solution. The second example problem is:

$$\begin{aligned}
 \text{Max } z^* &= 500x_1 + 900x_2 + 300x_3 + 600x_4 + 100x_5 + 300x_6 \\
 \text{st. } & -x_1 + 14x_2 - 5x_3 + 7x_4 + 5x_5 + 6x_6 \leq 15 \\
 & 9x_1 + 9x_2 + 6x_3 - 7x_4 + 9x_5 + 11x_6 \leq 25 \\
 & 12x_1 + 15x_2 + 20x_3 + 17x_4 - 8x_5 + 13x_6 \leq 35 \\
 & 5x_1 + 8x_2 + 2x_3 + 8x_4 + 4x_5 + 9x_6 \leq 15 \\
 & 8x_1 + 15x_2 + 5x_3 + 12x_4 + 9x_5 + 14x_6 \leq 30 \\
 & x_i \geq 0 \quad \text{for } i = 1, 2, 3, 4, 5, 6
 \end{aligned}$$

Following the same algorithm, the problem is solved as follows:

Step 1. Input the problem.

Step 2 and 3. Add one variable for each variable in the original problem. Additionally, add one integer sum constraint for each nonzero variable in the objective function. The problem should now appear as follows:

$$\begin{aligned}
 \text{Max } z^* &= 500x_1 + 900x_2 + 300x_3 + 600x_4 + 100x_5 + 300x_6 + 0x_7 + 0x_8 + 0x_9 + 0x_{10} + 0x_{11} + 0x_{12} \\
 \text{st. } & -x_1 + 14x_2 - 5x_3 + 7x_4 + 5x_5 + 6x_6 \leq 15 \\
 & 9x_1 + 9x_2 + 6x_3 - 7x_4 + 9x_5 + 11x_6 \leq 25 \\
 & 12x_1 + 15x_2 + 20x_3 + 17x_4 - 8x_5 + 13x_6 \leq 35 \\
 & 5x_1 + 8x_2 + 2x_3 + 8x_4 + 4x_5 + 9x_6 \leq 15
 \end{aligned}$$

$$8x_1 + 15x_2 + 5x_3 + 12x_4 + 9x_5 + 14x_6 \leq 30$$

$$\begin{aligned} x_1 &+ x_7 = 1 \\ &x_2 &+ x_8 = 1 \\ &&x_3 &+ x_9 = 1 \\ &&&x_4 &+ x_{10} = 1 \\ &&&&x_5 &+ x_{11} = 1 \\ &&&&&x_6 + x_{12} = 1 \\ x_i &\geq 0 \quad \text{for } i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \end{aligned}$$

Step 4. Solve the LP.

Final Tableau after initial LP

Basis	x_5	x_6	x_7	x_8	S3	S4	RHS
S1	-5.571	-4.5	4.786	-6.143	0.429	-1.786	1.857
S2	20.619	21.889	-12.698	-16.984	-0.492	1.921	6.905
x_3	-1.048	-0.389	-0.087	0.127	0.064	-0.135	0.238
x_4	0.762	1.222	-0.603	-1.032	-0.016	0.159	0.191
S5	5.095	1.278	-0.325	-3.254	-0.127	-1.23	3.524
x_1	0	0	1	0	0	0	1
x_2	0	0	0	1	0	0	1
x_9	1.048	0.389	0.087	-0.127	-0.064	0.135	0.762
x_{10}	-0.762	-1.222	.603	1.032	0.016	-0.159	0.81
x_{11}	1	0	0	0	0	0	1
x_{12}	0	1	0	0	0	0	1
Red. Cost	-42.857	-316.667	-111.905	-319.048	-9.524	-54.762	1585.714

Figure 3.6 Example 2 Tableau MCP Trial (0)

Step 5. Record the feasible solution. The variables in the basis are $x_1^{(0)}=1, x_2^{(0)}=1, x_3=.238, x_4^{(0)}=.191, x_9^{(0)}=.762, x_{10}^{(0)}=.81, x_{11}^{(0)}=1, x_{12}^{(0)}=1,$ and $z^{(0)}=1585.714.$

Step 6. Check the variables against the tolerance; in this problem it is .9. Since all of the variables are not within the tolerance, proceed to step 7.

Step 7. Calculate the profit gradients. Using the equation 12 again returns

$$r_1^{(0)} = M$$

$$r_2^{(0)} = M$$

$$r_3^{(0)} = 40.894$$

$$r_4^{(0)} = 185.58$$

$$r_5^{(0)} = 42.857$$

$$r_6^{(0)} = 316.667$$

$$r_7^{(0)} = 111.905$$

$$r_8^{(0)} = 319.048$$

$$r_9^{(0)} = 148.813$$

$$r_{10}^{(0)} = 56.243$$

$$r_{11}^{(0)} = M$$

$$r_{12}^{(0)} = M$$

Step 8. Calculate the profit bounds. Using the same formula for calculating profit bounds leads to

$$\phi_1^{(0)} = 1585.714 - (1-1)M = 1585.714$$

$$\phi_2^{(0)} = 1585.714 - (1-1)M = 1585.714$$

$$\phi_3^{(0)} = 1585.714 - (1-.238)40.894 = 1554.553$$

$$\phi_4^{(0)} = 1585.714 - (1-.191)185.58 = 1435.58$$

$$\phi_5^{(0)} = 1585.714 - 42.857 = 1542.857$$

$$\phi_6^{(0)} = 1585.714 - 316.667 = 1269.047$$

$$\phi_7^{(0)} = 1585.714 - 111.905 = 1473.809$$

$$\phi_8^{(0)} = 1585.714 - 319.048 = 1266.666$$

$$\phi_9^{(0)} = 1585.714 - (1-.762)148.813 = 1550.297$$

$$\phi_{10}^{(0)} = 1585.714 - (1-.81)56.243 = 1575.028$$

$$\phi_{11}^{(0)} = 1585.714 - (1-1)M = 1585.714$$

$$\phi_{12}^{(0)} = 1585.714 - (1-1)M = 1585.714$$

Step 9. Build the profit constraints by using equation 11.

$$\begin{aligned}
 -1085.714x_1 + 900x_2 + 300x_3 + 600x_4 + 100x_5 + 300x_6 - 1473.809x_7 &\leq 0 \\
 500x_1 - 685.714x_2 + 300x_3 + 600x_4 + 100x_5 + 300x_6 - 1266.666x_8 &\leq 0 \\
 500x_1 + 900x_2 - 1254.553x_3 + 600x_4 + 100x_5 + 300x_6 - 1550.297x_9 &\leq 0 \\
 500x_1 + 900x_2 + 300x_3 - 835.58x_4 + 100x_5 + 300x_6 - 1575.028x_{10} &\leq 0 \\
 500x_1 + 900x_2 + 300x_3 + 600x_4 - 1442.857x_5 + 300x_6 - 1585.714x_{11} &\leq 0 \\
 500x_1 + 900x_2 + 300x_3 + 600x_4 + 100x_5 - 969.047x_6 - 1585.714x_{12} &\leq 0
 \end{aligned}$$

Step 10. Rebuild the LP and return to step 4. As in the first example, the new LP will be the same as the original with the additional profit constraints.

The process is continued with the following results during each MCP trial:

MCP TRIAL	VARIABLES	SOLUTION
1	$x_1=.849, x_2=1, x_3=.639, x_5=.37, x_7=.152, x_9=.361, x_{10}=1, x_{11}=.63, x_{12}=1$	$z=1552.904$
2	$x_1=.999, x_2=1, x_3=.177, x_4=.107, x_5=.01, x_6=.004, x_9=.823, x_{10}=.893, x_{11}=.902, x_{12}=.956$	$z=1540.346$
3	$x_1=.999, x_2=1, x_3=.196, x_4=.007, x_5=.13, x_6=.005, x_9=.804, x_{10}=.93, x_{11}=.87, x_{12}=.952$	$z=1526.244$
4	$x_1=1, x_2=.999, x_3=.162, x_4=.008, x_5=.107, x_6=.004, x_9=.838, x_{10}=.924, x_{11}=.893, x_{12}=.963$	$z=1515.945$
5	$x_1=1, x_2=1, x_3=.181, x_4=.006, x_5=.009, x_6=.003, x_9=.819, x_{10}=.94, x_{11}=.91, x_{12}=.971$	$z=1508.333$
6	$x_1=.999, x_2=.999, x_3=.161, x_4=.006, x_5=.082, x_6=.002, x_9=.839, x_{10}=.936, x_{11}=.918, x_{12}=.976$	$z=1502.192$
7	$x_1=1, x_2=.999, x_3=.177, x_4=.005, x_5=.006, x_6=.002, x_9=.823, x_{10}=.946, x_{11}=.934, x_{12}=.981$	$z=1497.703$
8	$x_1=.885, x_2=1, x_3=.49, x_5=.005, x_7=.115, x_9=.51, x_{10}=.999, x_{11}=.948, x_{12}=.999$	$z=1494.516$
9	$x_1=1, x_2=1, x_3=.107, x_4=.007, x_5=.008, x_6=.02, x_9=.983, x_{10}=.926, x_{11}=.925, x_{12}=.98$	$z=1490.123$
10	$x_1=1, x_2=.999, x_3=.13, x_4=.005, x_5=.102, x_6=.003, x_9=.87, x_{10}=.954, x_{11}=.898, x_{12}=.975$	$z=1484.384$
11	$x_1=1, x_2=1, x_3=.1, x_4=.006, x_5=.009, x_6=.002, x_9=.9, x_{10}=.942, x_{11}=.912, x_{12}=.98$	$z=1479.84$
12	$x_1=.999, x_2=.999, x_3=.119, x_4=.045, x_5=.008, x_6=.002, x_9=.881, x_{10}=.955, x_{11}=.919, x_{12}=.983$	$z=1476.065$
13	$x_1=.999, x_2=1, x_3=.1, x_4=.005, x_5=.008, x_6=.002, x_9=.899, x_{10}=.949, x_{11}=.923, x_{12}=.985$	$z=1472.851$
14	$x_1=.999, x_2=1, x_3=.112, x_4=.004, x_5=.007, x_6=.001, x_9=.886, x_{10}=.957, x_{11}=.931, x_{12}=.988$	$z=1470.196$
15	$x_1=1, x_2=1, x_3=.101, x_4=.046, x_5=.007, x_6=.001, x_9=.899, x_{10}=.954, x_{11}=.935, x_{12}=.989$	$z=1467.88$
16	$x_1=1, x_2=1, x_3=.109, x_4=.004, x_5=.058, x_6=.001, x_9=.891, x_{10}=.959, x_{11}=.942, x_{12}=.991$	$z=1465.967$
17	$x_1=1, x_2=1, x_3=.101, x_4=.004, x_5=.006, x_6=.001, x_9=.899, x_{10}=.957, x_{11}=.944, x_{12}=.992$	$z=1464.27$
18	$x_1=1, x_2=.999, x_3=.108, x_4=.004, x_5=.005, x_6=.001, x_9=.892, x_{10}=.961, x_{11}=.951, x_{12}=.993$	$z=1462.873$
19	$x_1=.793, x_2=1, x_3=.537, x_5=.003, x_7=.207, x_9=.463, x_{10}=1, x_{11}=.968, x_{12}=.999$	$z=1460.802$

Figure 3.7 Solutions for Example 2

After step 4 of MCP trial 20, proceed to step 5.

Step 5. Record the feasible solution.

$$x_1^{(20)} = .999$$

$$x_2^{(20)} = 1$$

$$x_3^{(20)} = .08$$

$$x_4^{(20)} = .05$$

$$x_5^{(20)} = .05$$

$$x_6^{(20)} = .007$$

$$x_9^{(20)} = .924$$

$$x_{10}^{(20)} = .95$$

$$x_{11}^{(20)} = .948$$

$$x_{12}^{(20)} = .993$$

Step 6. Do all variables satisfy the tolerance? Yes, they do; then go to step 11.

Step 11. Force the variables to their respective values. Thus the variables now become $x_1=1, x_2=1, x_3=0, x_4=0, x_5=0, x_6=0$, with an optimal objective function value of $z^*=1400$.

Stop.

These examples are not very complex but they give an insight as to what MCP is doing and the steps necessary to find the solution to this type of problem. Clearly, many steps are required to enable MCP to find the solution to a problem and therefore some code is required.

3.6 Proof for the Profit Constraints:

The profit constraints are the driving force behind MCP. They are what allows the algorithm to take successive cuts from the solution space and drive the problem to an optimal 0-1 solution. A question that exists is: “Why don’t the cuts that form the profit constraints cut out a lattice point that is a potential solution?” The following short proof demonstrates why the constraints work and that they will not drive the solution below its optimal 0-1 solution. Additional details can be found in Dr. Healy’s original proof found in reference 10.

Suppose that there are integer sets of x ’s, y ’s, z ’s, etc., and that at optimality one x , one y , one z , etc. are equal to 1. Now let Γ be the first trial at which any profit bound wrongly drives ϕ below z^* . Furthermore, suppose the erroneous bound has the coefficient $\phi_\alpha^{(\Gamma)}$ for x_α in the x profit constraint. This presents two possible cases to consider.

Case 1.

If $x_\alpha^{(\Gamma)}=1$, then $z^* > \phi_\alpha^{(\Gamma)} = \phi^{(\Gamma)}$. Then at trial $\Gamma-1$, the problem would be $\phi_\alpha^{(\Gamma-1)} \geq z^*$, $\phi_\beta^{(\Gamma-1)} \geq z^*$, for all profit constraints from the definition of Γ . Thus the MCP solution (z^*) is a legitimate LP solution at trial Γ and therefore must be $\phi^{(\Gamma)} \geq z^*$, which is a contradiction.

Case 2.

In the second case, we have $x_\alpha^{(\Gamma)} < 1$, which leads to $z^* > z^{(\Gamma)} - (1 - x_\alpha^{(\Gamma)})r_\alpha^{(\Gamma)}$, or $r_\alpha^{(\Gamma)} > (z^{(\Gamma)} - z^*) / (1 - x_\alpha^{(\Gamma)})$; this, however, violates the definition of the profit gradient ($r_\alpha^{(\Gamma)}$). To demonstrate the claim above, it must be shown that $r_j^{(i)}$ as was defined earlier in this chapter is a lower bound to $(z^{(i)} - z^*) / (1 - x_j^{(i)})$ or $x_j^{(i)} < 1$. This can be done by showing that $r_j^{(i)}$ is exactly the minimum rate of change of z with respect to x_j at $x_j^{(i)}$. Consequently, it can be seen that r_j is a nondecreasing function of x_j . Suppose x_j is a basic variable and occupies the i th row. If x_k (variable in column k) is introduced into the basis, pivoting in rows, then x_j will increase only if $a_{ik}^{(i)} < 0$. The decrease in the objective function will be $b_s^{(i)} a_{0k}^{(i)} / a_{sk}^{(i)}$, the increase in x_j will be $-b_s^{(i)} a_{ik}^{(i)} / a_{sk}^{(i)}$, and the ratio is $a_{0k}^{(i)} / a_{ik}^{(i)}$ as described in equation 12, which is the amount z decreases with respect to an increase in x_j . If all $a_{ik}^{(i)} \geq 0$, then x_j cannot be increased and $r_j^{(i)}$ is assigned a large positive value. If x_j is a nonbasis variable in column k , then $a_{0k}^{(i)}$ is the minimum amount z will decrease for an increase in x_j . Hence it follows that

$$r_j^{(i)} = \begin{cases} a_{0k}^{(i)} & \text{if } x_j \text{ is non basic and occupies column } k \\ \min_{a_{ik}^{(i)} < 0} \frac{a_{0k}^{(i)}}{a_{ik}^{(i)}} & \text{if } x_j \text{ is basic and occupies row } i, \text{ take the minimum ratio} \\ & \text{using the negative entries of the row and deleting all } S_j \text{ columns} \end{cases}$$

is the profit gradient that determines the fraction of the amount that the objective function can change. This change therefore is used to determine the profit bound which will not allow the profit constraint to drive the solution below its MCP optimal.

Chapter 4

RESULTS OF TEST PROBLEMS**4.1 Discussion:**

In order to validate MCP as a viable alternative integer programming code, seventeen test problems have been included in Appendix A. These problems are a mixture of capital budgeting, knapsack, and shift scheduling problems taken from a variety of sources. Knapsack problems and shift scheduling are similar to capital budgeting problems in that they are also 0-1 integer programming problems. In both cases, a person is selected to work on a particular shift or is not selected or an item is selected to be used or is not selected to be used.

Knapsack problems are problems in which there are many items to select to be transported but there is only a limited capacity on the vehicle. In this case, the user must select which items are the most profitable to be transported [23]. Consider loading a semi truck with N items. Each item has a weight (w_i) and a value (v_i) for $i= 1,2,\dots,N$. The maximum weight the truck can hold is W . The user must therefore determine which is the most valuable cargo to be transported in order to maximize the return on investment. These problems are of the same form as capital budgeting problems.

Shift scheduling problems are problems in which the user is trying to minimize the number of workers to put on a particular shift so that his/her cost is minimized while still performing up to a specified level. For example, a grocery store will typically have more employees working in the afternoon and evening hours than during morning or night hours. Management knows that many people will arrive at the store on the way home from work to pick up food for dinner so must plan the shifts accordingly.

Shift scheduling problems are not of the same form as capital budgeting problems; however, they are minimization problems and so they can be formulated as:

$$\begin{aligned} \text{Minimize} \quad & z^* = \sum_{j=1}^n c_j x_j \\ \text{subject to:} \quad & \sum_{j=1}^n a_{ij} x_j \geq b_i \quad \text{for } i=1,2,\dots,m; \quad a_{ij} \geq 0 \quad \forall i, \forall j \\ & x_j \geq 0 \quad \text{for } j=1,2,\dots,n. \end{aligned} \quad (\text{eq 14})$$

Since they are minimization problems, they must be reformulated so that they become maximization problems. This is done by using a ones-complement transformation. This procedure rewrites each of the variables as its complement and subtracts the sum of the coefficients of the objective function from the objective function. For example, consider the problem of the following form:

$$\begin{aligned}
 &\text{Minimize } z^* = x_2 + 4x_3 \\
 &\text{st} \quad -2x_1 + 4x_3 \geq 1 \\
 &\quad \quad x_1 + x_2 + x_3 \geq 2 \\
 &\quad \quad x_i \geq 0 \quad \text{for } i=1,2,3
 \end{aligned}$$

This problem can be rewritten as a maximization problem by letting:

$$x_1 = 1 - y_1; \quad x_2 = 1 - y_2; \quad x_3 = 1 - y_3$$

If the variables are substituted into the original problem and the sum of the objective function coefficients are subtracted from the new objective function, it yields

$$\begin{aligned}
 &\text{Maximize } z^* = y_2 + 4y_3 - 5 \\
 &\text{st} \quad -2y_1 + 4y_3 \leq 1 \\
 &\quad \quad y_1 + y_2 + y_3 \leq 1 \\
 &\quad \quad y_i \geq 0; \quad \text{for } i=1,2,3
 \end{aligned}$$

This problem is then run on MCP as any other problem; however, when the solution is obtained for the maximization problem, it must then be substituted into the equations $x_1 = 1 - y_1$, $x_2 = 1 - y_2$, and $x_3 = 1 - y_3$ to obtain the solution to the minimization problem.

4.2 General Results:

Seventeen test problems were run using the code found in Appendix B. Fourteen of these test problems were solved correctly using MCP. These problems ranged from 10 variables and 1 constraint to 15 variables and 50 constraints in size. The results are listed in the following table (in all cases, STORM gave the correct solution). All of the problems were run on a 486DX computer with a speed of 33 MHz.

Results of Test Problems

Prob.	Type/ Size	MCP Soln (Vbls that = 1)	# Trials	MCP Time	STORM Soln (Vbls that = 1)	# Nodes	STORM Time
1	CB 5x7	x_2, x_3, x_4, x_7 $z^*=766$	17	2 min 31 sec	x_2, x_3, x_4, x_7 $z^*=766$	6	10 sec
2	CB 6x8	x_1, x_2, x_4, x_6, x_8 $z^*=2033$	84	25 min 1 sec	x_1, x_2, x_4, x_6, x_8 $z^*=2033$	14	10 sec
3	CB 9x9	x_2, x_5, x_6, x_7 $z^*=3724$	59	14 min 33 sec	x_2, x_5, x_6, x_7 $z^*=3724$	30	12 sec
4	CB 10x6	x_2, x_3, x_6 $z^*=3800$	4	30 sec	x_2, x_3, x_6 $z^*=3800$	16	10 sec
5	CB 10x15	$x_1, x_2, x_4, x_6, x_7,$ $x_9, x_{10}, x_{14}, x_{15}$ $z^*=4015$	11	8 min 50 sec	$x_1, x_2, x_4, x_6, x_7,$ $x_9, x_{10}, x_{14}, x_{15}$ $z^*=4015$	110	25 sec
6	CB 20x5	$x_1, x_{10}, x_{14}, x_{15},$ $x_{16}, x_{17}, x_{18},$ x_{19}, x_{20} $z^*=6120$	6	6 min 5 sec	$x_1, x_{10}, x_{14}, x_{15},$ $x_{16}, x_{17}, x_{18},$ x_{19}, x_{20} $z^*=6120$	310	44 sec
7	KS 1x10	$x_3, x_5, x_6, x_7, x_8,$ $z^*=50$	91	23 min 28 sec	x_3, x_5, x_6, x_7, x_8 $z^*=50$	8	9 sec
8	KS 1x10	$x_3, x_5, x_6, x_7, x_8,$ x_9, x_{10} $z^*=52$	25	6 min 42 sec	$x_3, x_5, x_6, x_7, x_8,$ x_9, x_{10} $z^*=52$	16	9 sec
9	KS 1x10	$x_3, x_4, x_5, x_7,$ x_8, x_9, x_{10} $z^*=57$	2	34 sec	$x_3, x_4, x_5, x_7,$ x_8, x_9, x_{10} $z^*=57$	14	10 sec
10	KS 1x10	$x_3, x_4, x_5, x_6, x_8,$ x_9, x_{10} $z^*=62$	1	18 sec	$x_3, x_4, x_5, x_6, x_8,$ x_9, x_{10} $z^*=62$	8	9sec
11	KS 1x10	$x_3, x_4, x_5, x_6, x_7,$ x_8, x_9, x_{10} $z^*=67$	1	16 sec	$x_3, x_4, x_5, x_6, x_7,$ x_8, x_9, x_{10} $z^*=67$	14	10 sec
12	KS 1x10	$x_4, x_5, x_6, x_8,$ $x_9, x_{10},$ $x_2=.0678$ $x_3=.9999$ $x_7=.9999$ $z^*=68.221$	62	17 min 10 sec	$x_2, x_4, x_5, x_6,$ x_7, x_8, x_9, x_{10} $z^*=68$	9	9 sec

CB denotes Capital Budgeting problem

KS denotes Knapsack problem

Figure 4.1 Results of Test Problems

Figure 4.1 Continued

Results of Test Problems

Prob.	Type/ Size	MCP Soln (Vbls that = 1)	# Trials	MCP Time	STORM Soln (Vbls that = 1)	# Nodes	STORM Time
13	KS 1x10	x_3, x_4, x_{10} $x_2=.295$ $x_5=.9999$ $x_6=.9999$ $x_7=.9999$ $x_8=.9999$ $x_9=.9999$ $z^*=72.3093$	32	6 min 15 sec	$x_2, x_4, x_5, x_6,$ x_7, x_8, x_9, x_{10} $z^*=68$	9	9 sec
14	KS 1x10	$x_3, x_4, x_6, x_7,$ x_9, x_{10} $x_2=.5115$ $x_5=.9999$ $x_8=.9999$ $z^*=76.207$	18	4 min 48 sec	$x_2, x_3, x_4, x_5,$ x_7, x_8, x_9, x_{10} $z^*=75$	13	10 sec
15	KS 1x10	$x_2, x_3, x_4, x_5,$ $x_6, x_7, x_8, x_9,$ x_{10} $z^*=85$	1	19 sec	$x_2, x_3, x_4, x_5,$ $x_6, x_7, x_8, x_9,$ x_{10} $z^*=85$	14	10 sec
16	SS 15x15	$x_5, x_6, x_7, x_8,$ $x_9, x_{10}, x_{11},$ x_{12}, x_{14}, x_{15} $z^*=10$	26	28 min 49 sec	$x_5, x_6, x_7, x_8,$ $x_9, x_{10}, x_{11},$ x_{12}, x_{14}, x_{15} $z^*=10$	45	28 sec
17	SS 50x15	$x_1, x_2, x_6, x_7,$ $x_9, x_{12}, x_{13},$ x_{14}, x_{15} $z^*=9$	3	6 min 17 sec	$x_1, x_2, x_3, x_4,$ $x_5, x_7, x_8,$ x_9, x_{10} $z^*=9$	68	1 min 2 sec

KS denotes Knapsack problem

SS denotes Shift Scheduling problem

As aforementioned, MCP solved 14 out of these 17 problems correctly. The three problems that were not solved correctly did not have any noticeable differences from the other problems with respect to formulation or variables. The problem appears to be with the LP solver in the code. For problems that require a large number of matrix inversions, it appears that round-off error continues to compound until such a point that the correct variables are no longer in the basis. This is due to the fact that the LP solver uses the simplex method and carries all of the entries whereas commercial LP solvers use methods of sparse matrices.

One noticeable difference between the problems is in the time required to solve them. For this analysis MCP was compared to STORM, a mathematical programming software package. STORM uses the method of branch and bound to solve the problems and is extremely fast in the computations. The LP portion of STORM is remarkably faster than the LP solver in MCP. This is important to MCP because the algorithm continuously solves LP problems until the solution falls within the tolerance set by the user. The MCP code quickly calculates the profit gradients and formulates the profit constraints; the majority of the time is spent solving the resulting LP problem. The LP solver in STORM usually requires only a few seconds to solve, with the rest of the time spent on the enumeration process.

Experience in solving MCP problems has shown that the tolerance set by the user is also extremely important. If the tolerance is set too loosely, for example .8 in some cases,

the problem will give a nonoptimal answer. On the other hand, if the tolerance is set too tightly, for example .999, it can potentially take several hours to solve. In some cases, a tolerance between .8-.9 will work, but a value between .9-.95 seems to work best.

Unfortunately, there is no readily available way of identifying which tolerance will work best on a particular problem by inspection; the only way is to select a tolerance level and try it.

The knapsack and shift scheduling problems also did well using MCP although they were not part of the original hypothesis. These particular types of problems can be viewed as special cases of capital budgeting problems with respect to their form and overall objective. Their objective is to select or not select particular items or people in order to maximize the users' return on investment. These problems only expand the types of problems that MCP has the capability of solving.

Chapter 5

CONCLUSIONS AND AREAS FOR FURTHER RESEARCH**5.1 Conclusions:**

Overall, MCP is able to solve capital budgeting problems and similar problems that fall into the 0-1 class of integer programming problems. The method offers an alternative to solving these problems; however, there is room for improvement in using this method. Branch and bound methods dominate the integer programming software world today, but branch and bound methods will predictably take longer to converge as the problems get larger. Cutting plane methods have the potential to be faster because they only operate on a series of linear programming steps. One problem with cutting plane methods, however, is that they do not always yield the optimal solution. Capital budgeting problems are types of problems that if the solution is close and can be attained in a reasonable amount of time, then the user would be willing to accept the answer. For example, if the user is dealing with millions of dollars and a solution can be attained within several thousand dollars of the optimal in a fraction of the time required by other methods, then he/she would probably be willing to accept that solution.

MCP has the potential to be more competitive with current integer programming software if it could be linked to a commercial LP solver. This would definitely speed up the processing time required and could prevent errors from compounding as is seen in the

present simplex method. This thesis demonstrates MCP's potential by attaining the optimal solution in most cases and a very close solution in the other cases.

5.2 Author's Contribution:

This thesis demonstrates the expanded capability of MCP to capital budgeting problems and some special cases of capital budgeting problems. It demonstrates a method of reformulating the problems so that they fit the requirements for MCP by adding "dummy" variables and additional integer sum constraints to the problems that are not normally associated with capital budgeting problems.

This thesis also has code that incorporates a matrix generator for capital budgeting problems so that the user does not have to be concerned with adding the additional variables and constraints. This portion of MCP is totally transparent to the user. The code is written in QBASIC and is "user friendly" so that it can be run on any personal computer that has DOS. No additional requirements are necessary.

5.3 Areas for Further Research:

There are several areas of research that can be investigated to further the capability of MCP. These include the tolerance set by the user, translating the code to C or C++ and linking it to a commercial LP solver, and generalizing the code so that it can run various types of MCP problems.

The tolerance set by the user is very important in running MCP. My research has shown that it determines whether the user will obtain the optimal solution, and determines the length of time necessary to obtain the optimal solution. If the code were to incorporate a method determining a good tolerance level based on the type of problem, it could alleviate the need to “guess” a level and to hope that the solution is, in fact, the optimal solution. Another method could be similar to STORM in that the user can determine in integer programming problems how close to optimality he/she wants to be. The problems run in this thesis are small compared to problems that are being run in industry today. Some problems in industry take 2 weeks to obtain a solution using a 486DX. STORM allows the user to accept a solution within $x\%$ of optimal; this cuts down on the time required for the program to run. MCP could benefit from this approach because the LP solver could quickly obtain an 80% or 90% optimal solution which could be faster than present branch and bound methods.

MCP could also be speeded up by running it in a different language, especially C or C++. These languages operate much more quickly than BASIC and in the research conducted during this thesis would be much easier to link with a commercial LP solver. Many LP software packages are now made to link with various languages but most all have the capability to link with C or C++. The only requirement is that the software package must be able to write the final tableau to a file that can in turn be used by the MCP code.

Another area of further research could include statistical analysis on the objective function coefficients for the projects. The coefficient is the amount of profit to be gained if the project is undertaken. There are situations where the life of the project could be several years. Therefore the organization is slowly receiving the profit over a given time period. There are outside influences that could affect this profit, these include inflation and a change in interest rates. The profit associated with these projects could then be thought of as random variables. One would have to collect data on a particular project and determine the best fit for the distribution. This could be followed by simulating the scenario to determine the amount of profit realized for a particular project. Finally, the problem could be inputted into MCP to be solved.

MCP can be used to solve various other types of problems which include fixed charge, fractionation and blending, and liquids and containers problems. It is now shown to work on capital budgeting problems as well, but the formulation is different for these types of problems. Since all use MCP, a generalized code could be developed so that the user could identify which type of problem they are trying to solve and then the code would do any reformulation necessary and run the problem.

It has been well documented in the IP literature that integer programming problems in excess of 100 0-1 variables are economically intractable [23]. This thesis has preliminary results to show that were MCP to be implemented using a commercial LP

code, such problems as above might be solved to defined optimality in an economic amount of time.

REFERENCES CITED

1. Balanski, M. 1965. The Multiple-Choice Knapsack Model. Management Science 12 (3): 253-313.
2. Beale, E. and Small R. 1965. Mixed Integer Programming by Branch and Bound Technique. Proceedings of the Third IFIP Congress 2: 450-451.
3. Balas, E. 1964. An Additive Method for Solving Mixed-Variables Programming Problems. Operations Research 13: 517-546.
4. Dantzig, G. B. 1960. On the Significance of Solving Linear Programming Problems with Some Integer Variables. Econometrica 28: 30-44.
5. Dantzig, Fulkerson, and Johnson. 1954. Solution of Large Scale Traveling Salesman Problem. Journals of the Operations Research Society of America 2: 393-410.
6. Davis, R., Kendrick, D., and Weitzman, M. 1971. A Branch and Bound Algorithm for Zero-One Mixed Integer Programming Problems. Operations Research 19 (4): 1036-1044.
7. Glover, F. 1965. A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem. Operations Research 12: 879-919.
8. Gomory, R. E. and Baumol, W. 1960. Integer Programming and Pricing. Econometrica 12 (3): 521-550.
9. Haldi, J. 25 Integer Programming Test Problems. Working Paper No. 43. Graduate School of Business, Stanford University.
10. Healy, W. C. Jr. 1964. Multiple Choice Programming. Operations Research 12 (1): 122-138.
11. _____. 1964. Operating Procedures with MCP Systems Tape. Technical Division, Ethyl Corporation Research Laboratories. Ferndale, Michigan.

12. _____. 1994. Private Conversation, Colorado School of Mines. July.
13. Henderson, G. V. and Mukherjee, T. K. 1987. The Capital Budgeting Process: Theory and Practice. Interfaces 17 (2): 78-90.
14. Kim, Tai-Yoo. 1983. Ph.D. Dissertation. A Reformation Algorithm for a Class of Capital Budgeting Investment Problems. Colorado School of Mines, Golden, Colorado .
15. Lawler, E. L. and Bell, M. D. 1966. A Method for Solving Discrete Optimization Problems. Operations Research 14: 1098-1112.
16. Little, Murty, Sweeny, and Caroline. 1963. An Algorithm for Traveling Salesman Problem. Operations Research 11: 972-989.
17. Mintzberg, H. 1976. The Structure of Unstructured Decision Process. Administrative Science Quarterly 21 (2): 246-275.
18. Petersen, C. C. 1967. Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R&D Projects. Management Science 13 (9): 737-750.
19. Plane, D. and McMillan C. Jr. Discrete Optimization. Englewood Cliffs, New Jersey: Prentice-Hall Inc.: 22-24.
20. Salkin, H. M. 1975. Integer Programming. Reading, Mass: Addison-Wesley Publishing Company: 165-249.
21. Schrage, L. 1991. LINDO User's Manual. San Fransisco, California. Scientific Press.
22. "STORM" . 1992. Version 2.0. Storm Software, Inc. Cleveland, Ohio.
23. Taha, H. A. 1987. Operations Research. New York: McMillan Publishing Company: 344-355.
24. Trauth, C. A. and Woolsey, R. E. D. 1969. Integer Linear Programming: A Study in Computational Efficiency. Management Science. 15(9): 481-493.

25. Woempner, M. 1994. Ph.D. Dissertation. Multiple Choice Programming. A Generalized Reduced Gradient Method for Solving Mixed Integer Problems with Zero-One and Continuous Variables. Colorado School of Mines, Golden Colorado.
26. Wu, N. and Coppins, R. 1981. Linear Programming and Extensions. New York: McGraw-Hill: 93-99.

Appendix A: Test Problems

This appendix gives the test problems run on a 486DX/33 MHz using MCP. The first problem is written out in the form that one would expect to see in an industrial setting.

Problem 1. Company XYZ is considering which of seven potential projects to do in the upcoming fiscal year. Its goal is to maximize the return on investment from these projects. The company accountants have determined that project 1 through 7 will yield \$180, \$190, \$195, \$183, \$178, \$187, and \$198 respectively. There are five constraints that must be considered. Presently, the accountants say there is a total of only \$90 available with which to do these projects. Each project will require \$15, \$12, \$19, \$6, \$18, \$23, and \$20 respectively to complete. The second constraint is man hours. The company foremen say there are only 76 man hours available for these projects. Additionally, the foremen have determined that each project will require 20, 5, 13, 9, 26, 20, and 12 man hours respectively. The third constraint, also determined by the foremen, concerns the new drill. The drill only has 55 hours free this upcoming fiscal year. Each project requires 9, 10, 18, 0, 7, 15, and 23 drilling hours respectively to complete. As is the case in many companies, politics is also involved. Higher management has determined a scale of importance for these projects is: 8, 6, 8, 6, 7, 12, and 14 respectively. The total of these cannot exceed 43. Lastly, the transportation people have

identified 40 trucks that are available to do these projects. Furthermore, they have determined each project will require 18, 11, 7, 10, 19, 16, and 7 trucks respectively to complete. Which projects should be done so that Company XYZ can realize its goal?

Problem 1

j	MAX	(all $b_j \leq$)				
j	c_j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	a_{5j}
1	180	15	20	9	8	18
2	190	12	5	10	6	11
3	195	19	13	18	8	7
4	183	6	9	0	6	10
5	178	18	26	7	7	19
6	187	23	20	15	12	16
7	198	20	12	23	14	7
$b_j =$		90	76	55	43	40

MCP SOLUTION

$$z^* = 766 \quad x_2 = 1, \quad x_3 = 1, \quad x_4 = 1, \quad x_7 = 1$$

TOLERANCE: .9

MCP Trials = 17

TIME: 2 MIN. 31 SEC

STORM SOLUTION

$$z^* = 766 \quad x_2 = 1, \quad x_3 = 1, \quad x_4 = 1, \quad x_7 = 1$$

NODES = 6

TIME: 10 SEC

Therefore, do projects 2, 3, 4, and 7 for a net return on investment of \$766

Problem 2

MAX (all $b_j \leq$)							
j	c_j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	a_{5j}	a_{6j}
1	560	40	16	38	8	38	33
2	300	10	41	32	30	30	27
3	620	30	16	71	60	42	37
4	431	20	23	26	18	9	18
5	68	3	4	5	6	7	8
6	322	9	12	30	31	21	29
7	103	15	15	14	17	18	10
8	420	21	22	8	31	38	17
<hr/> $b_j =$		105	117	140	130	140	127

MCP Solution

$z^* = 2033, x_1 = 1, x_2 = 1, x_4 = 1, x_6 = 1, x_8 = 1$

TOLERANCE : .95

MCP TRIALS = 84

TIME : 25 MIN 1 SEC

STORM Solution

$z^* = 2033, x_1 = 1, x_2 = 1, x_4 = 1, x_6 = 1, x_8 = 1$

NODES = 14

TIME : 10 SEC

Problem 3

MAX (all $b_j \leq$)										
j	c_j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	a_{5j}	a_{6j}	a_{7j}	a_{8j}	a_{9j}
1	560	40	16	38	8	38	27	33	40	20
2	1125	91	92	39	71	52	48	67	85	45
3	300	25	41	32	30	30	25	40	35	18
4	620	30	16	71	60	42	53	48	56	33
5	2100	160	150	80	200	170	135	149	190	110
6	431	20	23	26	18	9	30	25	24	15
7	68	3	4	5	6	7	6	4	8	4
8	328	12	27	40	30	20	23	31	35	17
9	47	5	6	8	4	6	5	6	7	3
$b_j =$		275	275	170	330	270	245	290	320	180

MCP Solution

$z^* = 3724$, $x_2 = 1$, $x_5 = 1$, $x_6 = 1$, $x_7 = 1$

TOLERANCE : .97

MCP TRIALS = 59

TIME : 14 MIN 33 SEC

STORM Solution

$z^* = 3724$, $x_2 = 1$, $x_5 = 1$, $x_6 = 1$, $x_7 = 1$

NODES = 30

TIME : 12 SEC

Problem 5 [18]

MAX (all $b_j \leq$)

j	c_j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	a_{5j}	a_{6j}	a_{7j}	a_{8j}	a_{9j}	a_{10j}
1	100	8	8	3	5	5	5	0	3	3	3
2	220	24	44	6	9	11	11	0	4	6	8
3	90	13	13	8	6	7	7	1	5	9	9
4	400	80	100	20	40	50	55	10	20	30	35
5	300	70	100	20	30	40	40	4	14	29	29
6	400	80	90	30	40	40	40	10	20	20	20
7	205	45	75	20	16	19	21	0	6	12	16
8	120	15	25	3	5	20	9	6	12	12	15
9	160	28	28	12	18	18	18	0	10	10	10
10	580	90	120	14	44	29	29	6	18	30	30
11	400	130	130	40	60	70	70	32	42	42	42
12	140	32	32	20	16	21	21	3	9	18	20
13	100	20	40	13	11	17	17	0	12	18	18
14	1300	120	160	20	30	30	35	70	100	110	120
15	650	40	40	5	25	25	25	10	20	20	20
$b_j =$		550	700	118	240	280	310	110	205	260	270

MCP SOLUTION

$z^* = 4015$, $x_1=1$, $x_2=1$, $x_4=1$, $x_6=1$, $x_7=1$, $x_9=1$, $x_{10}=1$, $x_{14}=1$, $x_{15}=1$

TOLERANCE: .9

MCP TRIALS = 11

TIME: 8 MIN 50 SEC

STORM SOLUTION

$z^* = 4015$, $x_1=1$, $x_2=1$, $x_4=1$, $x_6=1$, $x_7=1$, $x_9=1$, $x_{10}=1$, $x_{14}=1$, $x_{15}=1$

NODES = 110

TIME: 25 SEC

Problem 6 [18]

MAX (all $b_{jj} \leq$)						
j	c_j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	a_{5j}
1	100	8	8	3	5	5
2	220	24	44	6	9	11
3	90	13	13	8	6	7
4	400	80	100	20	40	50
5	300	70	100	20	30	40
6	400	80	90	30	40	40
7	205	45	75	20	16	19
8	120	15	25	3	5	20
9	160	28	28	12	18	18
10	580	90	120	14	44	29
11	400	130	130	40	60	70
12	140	32	32	20	16	21
13	100	20	40	13	11	17
14	1300	120	160	20	30	30
15	650	40	40	5	25	25
16	320	30	60	13	10	10
17	480	20	55	5	13	25
18	80	6	10	13	5	5
19	60	3	6	5	1	1
20	2550	180	240	20	80	100
$b_j =$		500	700	110	215	250

MCP SOLUTION

$z^* = 6120$, $x_1=1$, $x_{10}=1$, $x_{14}=1$, $x_{15}=1$, $x_{16}=1$, $x_{17}=1$,
 $x_{18}=1$, $x_{19}=1$, $x_{20}=1$

TOLERANCE : .8

MCP TRIALS = 6

TIME: 6 MIN 5 SEC

STORM SOLUTION

$z^* = 6120$, $x_1=1$, $x_{10}=1$, $x_{14}=1$, $x_{15}=1$, $x_{16}=1$, $x_{17}=1$,
 $x_{18}=1$, $x_{19}=1$, $x_{20}=1$

NODES = 310

TIME: 44 SEC

Problem 7-15 (Knapsack Problems) [12]

MAX

20 18 17 15 15 10 5 3 1 1

st

30 25 20 18 17 11 5 2 1 1 ≤ b_j
 $x_i \leq 1$ for $i = 1, 2, \dots, 10$ z_c

where b_j is:

Problem	1	2	3	4	5	6	7	8	9
b_j :	55	60	65	70	75	80	85	90	100

MCP SOLUTION PROBLEM #7

$z^* = 50$, $x_3=1$, $x_5=1$, $x_6=1$, $x_7=1$, $x_8=1$

TOLERANCE: .98

MCP TRIALS = 91

TIME: 23 MIN 28 SEC

STORM SOLUTION

$z^* = 50$, $x_3=1$, $x_5=1$, $x_6=1$, $x_7=1$, $x_8=1$

NODES = 8

TIME: 9 SEC

MCP SOLUTION PROBLEM #8 $z^* = 52, x_3=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

TOLERANCE: .9

MCP TRIALS = 25

TIME: 6 MIN 42 SEC

STORM SOLUTION $z^* = 52, x_3=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 16

TIME: 9 SEC

MCP SOLUTION PROBLEM #9 $z^* = 57, x_3=1, x_4=1, x_5=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

TOLERANCE: .9

MCP TRIALS = 2

TIME: 34 SEC

STORM SOLUTION $z^* = 57, x_3=1, x_4=1, x_5=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 14

TIME: 10 SEC

MCP SOLUTION PROBLEM #10

 $z^* = 62, x_3=1, x_4=1, x_5=1, x_6=1, x_8=1, x_9=1, x_{10}=1$

TOLERANCE: .93

MCP TRIALS = 1

TIME: 18 SEC

STORM SOLUTION

 $z^* = 62, x_3=1, x_4=1, x_5=1, x_6=1, x_8=1, x_9=1, x_{10}=1$

NODES = 8

TIME: 9 SEC

MCP SOLUTION PROBLEM #11

 $z^* = 67, x_3=1, x_4=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

TOLERANCE: .9

MCP TRIALS = 1

TIME: 16 SEC

STORM SOLUTION

 $z^* = 67, x_3=1, x_4=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 14

TIME: 10 SEC

MCP SOLUTION PROBLEM #12

 $z^* = 68.221, x_4=1, x_5=1, x_6=1, x_8=1, x_9=1, x_{10}=1, x_2=.0678$
 $x_3=.9999, x_7=.9999$

TOLERANCE: .9

MCP TRIALS = 62

TIME: 17 MIN 10 SEC

STORM SOLUTION

 $z^*=68, x_2=1, x_4=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 9

TIME: 9 SEC

MCP SOLUTION PROBLEM #13

$z^* = 72.3093, x_3=1, x_{10}=1, x_2=.295, x_5=.9999, x_6=.9999,$
 $x_7=.9999, x_8=.9999, x_9=.9999$

TOLERANCE: .9

MCP TRIALS = 32

TIME: 6 MIN 15 SEC

STORM SOLUTION

$z^*=68, x_2=1, x_4=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 9

TIME: 9 SEC

MCP SOLUTION PROBLEM #14

$z^* = 76.207, x_3=1, x_4=1, x_6=1, x_7=1, x_9=1, x_{10}=1, x_2=.5115$
 $x_5=.9999, x_8=.9999$

TOLERANCE: .9

MCP TRIALS = 18

TIME: 4 MIN 48 SEC

STORM SOLUTION

$z^*=75, x_2=1, x_3=1, x_4=1, x_5=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 13

TIME: 10 SEC

MCP SOLUTION PROBLEM #15

$z^* = 85, x_2=1, x_3=1, x_4=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

TOLERANCE: .9

MCP TRIALS = 1

TIME: 19 SEC

STORM SOLUTION

$z^* = 85, x_2=1, x_3=1, x_4=1, x_5=1, x_6=1, x_7=1, x_8=1, x_9=1, x_{10}=1$

NODES = 14

TIME: 10 SEC

Appendix B: MCP Code

This appendix contains the code that accompanies the MCP algorithm. It is written in QBASIC so any personal computer that has DOS version 5.0 or later can use this code. There are two programs that work together. The first is the matrix generator that puts the problem in the correct format for the MCP requirements as discussed in chapter 3. The second is the actual code that calculates the solution to the problem. Both codes are relatively easy to use and are interactive to allow the user to make several decisions. If the user is unfamiliar with MCP, it is best to go with the defaults that are built into the matrix generator.

The code for the matrix generator is listed below.

```

DECLARE SUB otpt ()

REM DATA1.BAS
CALL otpt
CLS
PRINT " YOU WILL FIRST ENTER THE NUMBER OF CONSTRAINTS,
VARIABLES, AND MAX MCP TRIALS": PRINT
INPUT " NUMBER OF CONSTRAINTS  ", NCON
INPUT " NUMBER OF VARIABLES  ", NVAR
INPUT " NUMBER OF MAXIMUM MCP TRIALS  ", NTRIAL
NROW = NCON + 2 * NVAR
NCOL = 2 * NVAR
WRITE #1, NROW, NCOL, NTRIAL
JCOST = NCOL + 3: ICOST = NROW + 3
DIM A(ICOST, JCOST), JROW(NCOL), ICOL(JROW)
CLS

```

rem the above information describes the size of the problem
 rem the following indicates the number and type of constraints

```
PRINT " YOU WILL NOW ENTER INFORMATION ABOUT THE CONSTRAINTS ":
PRINT
10 INPUT " NUMBER OF <= CONSTRAINTS  ", LE
INPUT " NUMBER OF >= CONSTRAINTS  ", GE
INPUT " NUMBER OF = CONSTRAINTS  ", EQ
IF (LE + GE + EQ - NCON = 0) THEN GOTO 20
PRINT "ERROR, TOTAL CONSTRAINTS DO NOT MATCH THOSE OF THE
PREVIOUS SCREEN, DO AGAIN ",
GOTO 10
```

```
rem this portion inputs all the rhs'
20 SLACK = 0
FOR I = 1 TO LE
  PRINT " FOR <= CONSTRAINT #"; I; : INPUT " INPUT THE RHS. "; RHS
  WRITE #1, I, RHS, SLACK
NEXT I
PRINT
FOR I = LE + 1 TO LE + GE
  PRINT " FOR >= CONSTRAINT #"; I; : INPUT " INPUT THE RHS. "; RHS
  WRITE #1, I, RHS, SLACK
NEXT I
PRINT
BIGM = -100
FOR I = LE + GE + 1 TO LE + GE + EQ
  PRINT " FOR = CONSTRAINT #"; I; : INPUT " INPUT THE RHS. "; RHS
  WRITE #1, I, RHS, BIGM
NEXT I
FOR I = 1 TO NVAR
  WRITE #1, NCON + I, 1, BIGM
NEXT I
FOR I = 1 TO NVAR
  WRITE #1, NCON + NVAR + I, 0, 0
NEXT I
CLS
```

rem this portion inputs the objective function coefficients

```

PRINT " YOU WILL NOW ENTER THE OBJECTIVE FUNCTION COEFFICIENTS
": PRINT
MAX = 100
FOR J = 1 TO NVAR
  VBL = 100
  PRINT " VARIABLE NUMBER "; VBL + J; : INPUT "OBJECTIVE FUNCTION
COEFFICIENT IS ", COEF
  WRITE #1, VBL + J, COEF
  IF COEFF > MAX THEN MAX = COEF
NEXT J
FOR I = NVAR + 1 TO NVAR * 2
  WRITE #1, VBL + I, 0
NEXT I
CLS

```

```

rem this portion deals with the constraint coefficients
PRINT "YOU WILL NOW ENTER THE COEFFICIENTS OF THE CONSTRAINTS."
PRINT "THIS WILL BE DONE IN THE FOLLOWING ORDER: <=, >=, AND =":
PRINT
VBL = 100
FOR I = 1 TO LE
  FOR J = 1 TO NVAR
    PRINT " FOR<= CONSTRAINT #"; I; "VARIABLE "; VBL + J; : INPUT "THE
COEFFICIENT IS ", COEFF
    IF COEFF < 0 THEN WRITE #1, I, VBL + J, COEFF
  NEXT J
  VBL = 100
PRINT
NEXT I
IF GE = 0 GOTO 50
PRINT " NOW ENTER THE >= CONSTRAINTS": PRINT
VBL = 100
FOR I = LE + 1 TO GE
  FOR J = 1 TO NVAR
    PRINT " FOR >= CONSTRAINT #"; I; "VARIABLE "; VBL + J; : INPUT "THE
COEFFICIENT IS ", COEFF
    IF COEFF < 0 THEN WRITE #1, I, VBL + J, COEFF
  NEXT J
  VBL = 100
PRINT

```

```

NEXT I
50 PRINT " NOW INPUT THE COEFFICIENTS FOR THE = CONSTRAINTS"
VBL = 100
FOR I = LE + GE + 1 TO EQ
  FOR J = 1 TO NVAR
    PRINT " FOR = CONSTRAINT #"; I; " VARIABLE "; VBL + J; : INPUT " THE
    COEFFICIENT IS ", COEFF
    IF COEFF <> 0 THEN WRITE #1, I, VBL + J, COEFF
  NEXT J
  VBL = 100
PRINT
NEXT I
VBL = 100
FOR I = NCON + 1 TO NCON + NVAR
  WRITE #1, I, VBL + I - NCON, 1
  WRITE #1, I, VBL + I - NCON + NVAR, 1
NEXT I
WRITE #1, 0, 0, 0
CLS

```

```

rem this portion handles the tolerance levels
PRINT " YOU WILL NOW ENTER THE TOLERANCE LEVELS. IF YOU WISH TO
GO WITH THE"
PRINT " DEFAULTS YOU CAN BYPASS THIS SECTION.": PRINT
PNALT = 10 * MAX
INPUT " DO YOU WANT THE PROGRAM DEFAULTS? (Y/N) "; YN$
IF UCASE$(YN$) = "Y" THEN GOTO 100
rem this is the part to finish; set up default measures
INPUT " TOLERANCE 1 ", TOL1
INPUT " TOLERANCE 2 ", TOL2
INPUT " A LARGE PHIJ ", PHIJ
INPUT " DEGENERACY FACTOR ", DGEN
INPUT " PERTM ", PERTM
WRITE #1, TOL1, TOL2, PHIJ, PNALT, DGEN, PERTM
GOTO 110
100 WRITE #1, .95, .7, 1000000, PNALT, .015, 1000
110 CLS

```

```

rem this is where the matrix generator constructs all of the integer sum constraints
rem and handles the relationship between integer sum constraints and profit constraints

```

```
FOR I = NCON + 1 TO NCON + NVAR
  WRITE #1, I, I + NVAR
NEXT I
WRITE #1, 0!, 0!
CLS
```

```
rem this part handles the relationship of the variables in the integer sum constraints
VBL = 100
FOR I = 1 TO NVAR
  WRITE #1, VBL + I, VBL + I + NVAR, NCON + I, 0
  WRITE #1, VBL + I + NVAR, VBL + I, NCON + I, 0
NEXT I
WRITE #1, -999, 0, 0, 0
CLOSE #1
END
```

```
SUB opt
rem select drive and file for the data.
CLS
INPUT "Write the data to drive (do NOT type the colon, type A, B, or C, etc.) ", d$
INPUT "The file name is (8 characters or less) ", nmf$
nfl$ = d$ + ":" + nmf$: PRINT nfl$
OPEN nfl$ FOR OUTPUT AS #1
END SUB
```

This portion of the code solves the problem as an MCP. In order to use this, however, a file must be created from the matrix generator first. This file is formatted in such a way as to allow this code to read the file. This, too, is interactive and allows the user to make several decisions. The code is as follows.

```

REM This code solves capital budgeting MCP problems
REM Use the matrix generator first to create a file for this code

4 CLS : PRINT "*****"
5 PRINT "BASIC TRANSLATION OF HEALY'S MCP PROGRAM, VERSION
CAPITAL BUDGETING"
6 PRINT "*****"

REM Input file name
INPUT "The drive the data is on is (A, B, or C, etc.) ", d$
INPUT "The file name is ", nmf$
namf$ = d$ + "." + nmf$
OPEN namf$ FOR INPUT AS #1
8 INPUT "DO YOU WANT SUCCESSIVE MCP TRIALS PRINTED OUT?(Y/N)"; yn$

10 REM read no of basic and non-basic variables and maximum trials
15 INPUT #1, nrow, ncol, maxtr
17 PRINT "ROWS="; nrow; " COLS="; ncol; " MAX TRIALS="; maxtr
18 jb = ncol + 1: jwork = ncol + 2: jcost = ncol + 3: IDJ = nrow + 1
20 iwork = nrow + 2: ICOST = nrow + 3: numtr = 0: lass = 1
DIM a!(ICOST, jcost), jrow(ICOST), ICOL(jcost), kflag(ncol), PHI!(ncol)
DIM nexcl(ncol, 4), intlo(ncol / 2, ncol + 2), n(ncol), kset(ncol / 2)
DIM B!(ICOST, jcost), svicol(jcost), svjrow(ICOST), svobj(jcost), r!(ICOST)

30 iter = 0: FOR j = 1 TO jcost: FOR i = 1 TO ICOST: a!(i, j) = 0!: NEXT i: NEXT j'
ZERO MATRIX
35 nrww = nrow - ncol / 2

```

```

REM this portion reads the variables from the file and puts in a matrix
REM read names of basic variables, rhs entries, and costs from 0001 consecutively
REM read names and costs of non-basic variables --- number from 0101.
REM read in matrix elements --- number rows and cols as above.
80 FOR i = 1 TO nrow: INPUT #1, jrow(i), a!(i, j), a!(i, jcost): NEXT i
90 FOR j = 1 TO ncol: INPUT #1, ICOL(j), a!(ICOST, j): svobj(j) = a!(ICOST, j)
NEXT j
100 INPUT #1, i, JAY, X: IF i <= 0 THEN GOTO 115
110 j = JAY - 100: a!(i, j) = X: GOTO 100
115 GOSUB 15000 ' GO AND SAVE THIS MATRIX FOR REUSE IN THE NEXT TRIAL
117 PRINT "MATRIX LOADED FOR TRIAL FOR MCP"
118 GOTO 120
119 IF UCASE$(yn$) = "Y" THEN GOSUB 14000

REM this inputs the tolerance levels set by the user
120 INPUT #1, tol1, tol2, phij, pnalt, dgen, pertm
125 PRINT "TOL1= "; tol1; " TOL2= "; tol2; " PHIJ= "; phij
127 PRINT "PNALT= "; pnalt; " DGEN= "; dgen; " PERTM= "; pertm
130 REM CREATE ID TABLE FOR SUMROW, CONSTRAINT, AND INTEGER VARIABLES
140 nvar = 0: i = 0
150 INPUT #1, IDSUM, IDCON
REM check for blank or negative trailer card
170 IF IDSUM <= 0! THEN GOTO 420
410 intlo(i + 1, 1) = IDSUM: intlo(i + 1, 2) = IDCON: i = i + 1: GOTO 150
420 intse = i
REM search starting matrix for integer variables
440 FOR i = 1 TO intse: kset(i) = 0!: M = 3
REM find sum row in matrix
460 INDEX = 1
470 IF (intlo(i, 1) - jrow(INDEX)) = 0! THEN GOTO 500
480 INDEX = INDEX + 1: IF INDEX <= nrow THEN GOTO 470
REM search sum row for 1.0 items
510 FOR j = 1 TO ncol
520 IF a!(INDEX, j) <> 1! THEN GOTO 540
530 intlo(i, M) = ICOL(j): M = M + 1: nvar = nvar + 1: kset(i) = kset(i) + 1
540 NEXT j
545 NEXT i

```

```

REM write output to allow user to check the problem
560 PRINT "MATRIX HAS "; intse; " SUM ROWS AND"; nvar; " INTEGER
VARIABLES"
570 nex = 0
580 i = 0
REM create table of exclusion sets, basic flags , and current phij
REM 0= non-basic, 1= basic variable
REM maximum number variables in exclusion set
620 INPUT #1, n(1), n(2), n(3), n(4)

REM check for blank or negative trailer card
REM blank cards indicate the end of the input file
640 IF n(1) <= 0 THEN GOTO 680
650 PHI!(i + 1) = phij
660 FOR j = 1 TO 4: nexcl(i + 1, j) = n(j): NEXT j
670 i = i + 1: nex = nex + 1: GOTO 620
680 PRINT nex; " EXCLUSION SETS SPECIFIED"
690 IF nex <> nvar THEN GOTO 710
700 GOTO 7000
710 PRINT "ERROR IN DATA"
715 STOP

REM routine for calculating the profit constraints
REM first profit gradients for basic variables
809 FOR i = 1 TO ncol: WH(i) = -1: r!(i) = -1!: NEXT i
810 FOR i = 1 TO nrow
  820 IF (jrow(i) < 100) THEN GOTO 890
  840 rhold = 1000000!
  845 FOR j = 1 TO ncol
    850 IF a!(i, j) >= 0! THEN GOTO 870
    855 ratio = -a!(IDJ, j) / a!(i, j)
    860 IF ratio >= rhold GOTO 870
    865 rhold = ratio
  870 NEXT j
  880 r!(jrow(i) - 100) = rhold' SET R!(J) AND CONTINUE
890 NEXT i

REM next find profit gradient for non-basic variables
910 FOR j = 1 TO ncol
  920 IF (ICOL(j) < 100) THEN GOTO 950

```

```

930 REM non-basic variable found in column j
940 r!(ICOL(j) - 100) = a!(IDJ, j)'SET R!(J) EQUAL TO SHADOW PRICE
950 NEXT j
REM print "r!(j)s":for i=1 to 12:print r!(i):next i

```

REM now calculate rprofit bounds

```

FOR j = 1 TO ncol: PHIOLD!(j) = PHI!(j): NEXT j
965 FOR j = 1 TO nrow
967 IF jrow(j) < 100 THEN GOTO 975
969 phin = a!(IDJ, jb) - (1 - a!(j, jb)) * r!(jrow(j) - 100)
970 IF (phin > PHI!(jrow(j) - 100)) THEN GOTO 975
973 PHI!(jrow(j) - 100) = phin
975 NEXT j
985 FOR i = 1 TO ncol
987 IF ICOL(i) < 100 THEN GOTO 995
991 phin = a!(IDJ, jb) - (1 - 0) * r!(ICOL(i) - 100)
992 IF (phin > PHI!(ICOL(i) - 100)) THEN GOTO 995
993 PHI!(ICOL(i) - 100) = phin
995 NEXT i

```

```

996 ' PRINT "PHI!(J)S"; TAB(20); "R!(J)s": FOR i = 1 TO ncol: PRINT PHI!(i);
TAB(22); r!(i): NEXT i 'PHIOLD!(I): NEXT I
997 GOTO 6930

```

REM this portion will print out the solution up to this point, it indicates the lp iterations,
 REM solution for the variables, and the number of mcp trials

```

3000 CLS : PRINT "L.P.OPTIMUM FOUND AT "; iter; " ITERATIONS AND "; numtr;
" MCP TRIALS"

```

```

3001 IF (lass = 3) THEN GOTO 3005
3002 REM INPUT T$
3003 IF UCASE$(yn$) = "N" THEN GOTO 3075
3005 PRINT "OPTIMUM VALUE OF OBJECTIVE FUNCTION IS"; a!(IDJ, jb)
3040 PRINT "NON-BASIC VARBL COST DJ"
3050 FOR j = 1 TO ncol' : IF j MOD 20 = 0 THEN GOSUB 20000
REM If you want to see program output then del the " ' " after ncol
3060 PRINT ICOL(j); TAB(17); a!(ICOST, j); TAB(25); a!(IDJ, j)
3070 NEXT j
3010 PRINT "BASIC VARIABLE COST BI"

```

REM this routine forces the variables to 0-1 if conditions are met

```

IF lass < 3 THEN GOTO 3020
FOR i = 1 TO nrow
  IF jrow(i) < 100 THEN GOTO 3015
  IF a!(i, jb) >= tol THEN a!(i, jb) = 1 ELSE a!(i, jb) = 0
3015 NEXT i

3020 FOR i = 1 TO nrow' : IF i MOD 20 = 0 THEN GOSUB 20000
REM if you want to stop program output del " ' " after nrow

3030 PRINT jrow(i); TAB(17); a!(i, jcost); TAB(25); a!(i, jb): NEXT i
3032 rlobj = 0!: FOR j = 1 TO nrow: IF (jrow(j) < 101) THEN GOTO 3036
3034 rlobj = rlobj + a!(j, jb) * sobj(jrow(j) - 100)
3036 NEXT j
REM GOSUB 20000 If you want to see output then del the rem and this sentence
3037 PRINT "LP Optimum found at "; iter; " iterations and "; numtr; " MCP trials"
3038 PRINT "REAL VALUE OF OBJECTIVE FUNCTION IS"; rlobj
3075 nrww = nrow ' SET NRWW TO SEACH ALL ROWS FOR REST OF TRIALS
3080 IF lass <= 2 THEN GOTO 7300
3090 PRINT "*****"
3092 PRINT "          END OF MCP JOB          "
      PRINT "*****"

3095 CLOSE #1
3096 END

REM the matrix is now rebuilt
6930 GOSUB 15100 ' read in original L.P. matrix & row and col id's
6937 IF UCASE$(yn$) = "N" THEN GOTO 6940
6938 PRINT 'rebuilt matrix before addn'l cost constraints is ": gosub 14000:input T$
6940 iter = 0

REM this portion calculates the additional constraints and adds them to the problem
REM update flags
7010 FOR i = 1 TO nex: kflag(i) = 0: NEXT i
REM assemble constraint elements
REM copy profit row
7040 FOR i = 1 TO intse: INDEX = intlo(i, 2): FOR j = 1 TO ncol: a!(INDEX, j) =
a!(ICOST, j): NEXT j: NEXT i
REM if phij non pos, read individual phi!(j),s
7060 IF phij > 0! THEN GOTO 7110

```

```
7080 FOR i = 1 TO nex: READ PHI!(i): NEXT i
7090 phij = 10!
REM subtract phi j for integer variables
7110 FOR i = 1 TO intse: KTR = kset(i) + 2
7120 k = 3
7130 j = 1
7140 IF intlo(i, k) <> nexcl(j, 1) THEN GOTO 7150
7145 GOTO 7160
7150 j = j + 1: IF j <= nex THEN GOTO 7140
7160 INDEX = intlo(i, 2)
7170 L = 1
7180 IF intlo(i, k) <> ICOL(L) THEN GOTO 7190
7185 GOTO 7200
7190 L = L + 1: IF L <= ncol THEN GOTO 7180
7200 a!(INDEX, L) = a!(INDEX, L) - PHI!(L)
7205 IF (ABS(a!(INDEX, L)) < .000001) THEN a!(INDEX, L) = 0!
7210 k = k + 1: IF k <= KTR THEN GOTO 7130
7215 NEXT i

REM the problem is now tested against the convergence criteria
REM jump to shadow price calculation and change problem class
7230 lass = 2: GOTO 8000'PRINT "AT 7230 LASS=";LASS: GOTO 8000
REM calculate shadow prices
REM test for convergence against tolerance
7340 numtr = numtr + 1
7350 IF maxtr > numtr THEN GOTO 7370
7360 tol = tol2: lass = 3: GOTO 7400
7370 tol = tol1: KOUNT = 0

7400 FOR i = 1 TO intse: KTR = kset(i) + 2
7410 FOR M = 3 TO KTR
7420 INDEX = 1
7430 IF intlo(i, M) - nexcl(INDEX, 1) = 0! THEN GOTO 7450
7440 INDEX = INDEX + 1: IF INDEX <= nex THEN GOTO 7430
7450 IF kflag(INDEX) > 0! THEN GOTO 7480
7460 ACT = 0!: REM PRINT INTLO(I,M),ACT
7470 GOTO 7700
7480 k = 1
7490 IF (intlo(i, M) - jrow(k)) = 0! THEN GOTO 7510
7500 k = k + 1: IF k <= nrow THEN GOTO 7490
```

```
7510 REM print intlo(i,m),a!(k,jb)
      IF (a(k, jb) >= .9999) THEN a(k, jb) = 1
      REM if (a(k, jb) < .0001) then a(k, jb) = 0
7520 IF (a!(k, jb) - tol) < 0! THEN GOTO 7700
7530 KOUNT = KOUNT + 1
      REM apply forcing cost
7550 a!(k, jcost) = a!(k, jcost) + pertm
7700 NEXT M
7710 NEXT i
      REM check for variable greater than tolerance in each set
7730 IF (KOUNT - intse = 0) THEN GOTO 7800
      REM remove forcing costs
7750 IF lass > 2 THEN GOTO 8010
7760 FOR i = 1 TO nrow
7770 a!(i, jcost) = a!(j, jcost) - pertm
7780 NEXT i
7790 GOTO 800 'PRINT "AT 7790":GOTO 5000 GO TO CALCULATE NEW PHIJ
      AND COMPARE
7800 lass = 3: PRINT " AT 7800 LASS SET TO LASS="; lass: GOTO 3000
      REM calculate shadow prices
8010 FOR j = 1 TO jb: a!(IDJ, j) = 0!
8020 FOR i = 1 TO nrow: a!(IDJ, j) = a!(i, j) * a!(i, jcost) + a!(IDJ, j): NEXT i
8030 a!(IDJ, j) = a!(IDJ, j) - a!(ICOST, j)
8040 NEXT j
      REM if yn$="n" then got to 8050, display option of matrix if wanted
8046 GOTO 8050
8047 PRINT "AT 8045": GOSUB 14000: INPUT T$

      REM this is the lp solver of the program
8050 REM solver
8060 REM determine variable in
8070 DTEST = 0!
8080 FOR j = 1 TO ncol
8085 IF a!(IDJ, j) >= 0! THEN GOTO 8110
8090 IF a!(IDJ, j) >= DTEST THEN GOTO 8110
8100 DTEST = a!(IDJ, j): jin = j
8110 NEXT j
8112 IF DTEST = 0! THEN GOTO 3000

8130 REM test for minimum pivot and determine variable out
```

```
8140 tphi = 1000: iout = 0
8145 i = 1
8147 IF (a!(i, jin) - .00000001#) <= 0! THEN GOTO 8300
8150 IF (a!(i, jb) < 0!) THEN GOTO 8178
8155 IF (a!(i, jb) = 0!) THEN GOTO 8180
8170 fhi = a!(i, jb) / a!(i, jin)
8173 REM if fhi < 0! then goto 8300
8174 IF (tphi - fhi) <= 0! THEN GOTO 8300
8176 tphi = fhi: iout = i: GOTO 8300
8177 fhi = 0!: iout = i: GOTO 8370' ZERO RIGHT HAND SIDE
8178 a!(i, jb) = 0!
8180 fhi = 0!: iout = i: GOTO 8370
8300 i = i + 1: IF i <= nrww THEN GOTO 8147
8310 IF iout <= 0 THEN GOTO 8340
8320 GOTO 8370
8330 INPUT T$
8340 PRINT "ERROR OR PROBLEM UNBOUNDED"
      PRINT "error occured in MCP trial "; numtr; "and lp iteration "; iter
      PRINT "jin is "; jin; "iout is "; iout
8350 STOP

8360 REM exchange names and costs of iout and jin, complete iteration
8375 piv = a!(iout, jin): i = 1
8380 IF (jrow(iout) - nexcl(i, 1)) <> 0 THEN GOTO 8410
8390 REM update flags on integer variables
8400 kflag(i) = 0: GOTO 8420
8410 i = i + 1: IF i <= nex THEN GOTO 8380
8420 i = 1
8430 IF (ICOL(jin) - nexcl(i, 1)) <> 0 THEN GOTO 8450
8440 kflag(i) = 1: GOTO 8460
8450 i = i + 1: IF i <= nex THEN GOTO 8430
8460 FOR i = 1 TO IDJ: a!(i, jwork) = a!(i, jin): a!(i, jin) = 0!: NEXT i
8470 a!(iout, jin) = 1!: FOR j = 1 TO jb: a!(iwork, j) = a!(iout, j): NEXT j
8480 ICOL(jwork) = ICOL(jin): ICOL(jin) = jrow(iout): jrow(iout) = ICOL(jwork)
8490 a!(ICOST, jwork) = a!(ICOST, jin): a!(ICOST, jin) = a!(iout, jcost)
      a!(iout, jcost) = a!(ICOST, jwork)
8500 FOR j = 1 TO jb: ratio = a!(iout, j) / piv
8510 IF ratio = 0! THEN GOTO 8600
8520 FOR i = 1 TO IDJ: a!(i, j) = a!(i, j) - ratio * (a!(i, jwork))
8530 REM test for essential zero
```

```

8535 IF (ABS(a!(i, j)) < .000001) THEN GOTO 8565
8540 IF (a!(i, j) + 1E-08) < 0! THEN GOTO 8570
8550 IF (a!(i, j) + 1E-08 = 0!) THEN GOTO 8565
8560 IF (a!(i, j) - 1E-08) > 0! THEN GOTO 8570
8565 a!(i, j) = 0!
8570 NEXT i
8600 NEXT j
8610 FOR j = 1 TO jb: a!(iwork, j) = a!(iwork, j) / piv: a!(iout, j) = a!(iwork, j): NEXT j
8620 FOR i = 1 TO ICOST: a!(i, jwork) = 0!: NEXT i
    REM IF lass = 3 THEN GOSUB 14000
8622 GOTO 8630
8623 IF UCASE$(yn$) = "N" THEN GOTO 8630
8625 PRINT "AT 8625 AFTER PIVOT": IF ((numtr <> 1) OR (iter < 60)) THEN GOTO
8630 ELSE GOSUB 14000: INPUT T$
8627 PRINT "AT 8627, KLFLAGS ARE"; kflag(1); kflag(2); kflag(3); kflag(4);
kflag(5); kflag(6)
8630 iter = iter + 1: GOTO 8070
REM this is the end of the lp solver

```

```

14000 LPRINT "real obj function is "; rlobj; TAB(35); "MATRIX NOW IS SHOWN
BELOW"; TAB(90); "RHS"

```

```

    LPRINT "MATRIX NOW IS SHOWN BELOW"; TAB(45); "MCP trial "; numtr; "
lp iteration "; iter
    LPRINT "JROW(I)"; TAB(12); ICOL(1); TAB(21); ICOL(2); TAB(31); ICOL(3);
TAB(41); ICOL(4); TAB(50); ICOL(5); TAB(59); ICOL(6); TAB(68); ICOL(7)
    FOR i = 1 TO ICOST
        LPRINT jrow(i); " ";
        LPRINT USING "##.##^"; a(i, 1); a(i, 2); a(i, 3); a(i, 4); a(i, 5); a(i, 6); a(i, 7)
    NEXT i

```

```

'14006 LPRINT "JROW(I)"; TAB(10); ICOL(1); TAB(20); ICOL(2); TAB(30);
ICOL(3); TAB(40); ICOL(4); TAB(50); ICOL(5); TAB(60); ICOL(6); TAB(70);
ICOL(7); TAB(80); ICOL(8); TAB(90); "RHS"
'14010 FOR i = 1 TO ICOST
'14021 LPRINT jrow(i); " ";
'14022 LPRINT USING "##.##^"; a(i, 1); a(i, 2); a(i, 3); a(i, 4); a(i, 5); a(i, 6);
a(i, 7); a(i, 8); a(i, 9); a(i, 10); a(i, 11)
'14030 NEXT i
    LPRINT "IOUT= "; iout; TAB(25); "jin= "; jin

```

14040 RETURN

REM this portion rebuilds the new matrix from the old matrix that has just been inverted from the lp solver

15000 REM matrix save routine for successive trials

15005 PRINT "SAVING MATRIX AND ROW AND COL ID'S FOR NEXT TRIAL"

15010 FOR j = 1 TO ncol + 3: svicol(j) = ICOL(j): NEXT j' SAVE COLUMN NAMES

15020 FOR i = 1 TO nrow + 3: svjrow(i) = jrow(i): NEXT i' SAVE ROW NAMES

15030 FOR j = 1 TO ncol + 3: FOR i = 1 TO nrow + 3: B!(i, j) = a!(i, j): NEXT i:
NEXT j

15040 RETURN

15100 REM matrix rebuild routine for successive trials

15105 REM print "building matrix and row and col id's for next trial"

15110 FOR j = 1 TO ncol + 3: ICOL(j) = svicol(j): NEXT j' REBUILD COLUMN
NAMES

15120 FOR i = 1 TO nrow + 3: jrow(i) = svjrow(i): NEXT i' REBUILD ROW NAMES

15130 FOR j = 1 TO ncol + 3: FOR i = 1 TO nrow + 3: a!(i, j) = B!(i, j): NEXT i:
NEXT j

15140 RETURN

18000 PRINT "YOU HAVE AN ERROR": STOP

20000 REM sub continue

LOCATE 24, 1

INPUT "Press <ENTER> to continue.", a\$

CLS

RETURN