AN INTEGER ZERO-ONE

CAPITAL BUDGETING

PROGRAMMING ALGORITHM

by

David E. Burch

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

T-4780

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 February 22, 1996

Signed: David E Burch
David E. Burch

Approved: Ruth A Maurer
Dr. Ruth A. Maurer
Thesis Advisor

Golden,Colorado

Date 2/22/96

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

ABSTRACT

In this thesis, a heuristic algorithm for solving a class of integer zero-one programming problems is presented. The algorithm solves zero-one maximization problems with less than or equal to constraints, or minimization problems with greater than or equal to constraints.

There are three key concepts used in the algorithm. First, it provides an effective method of computing payoff-per-unit-of-combined-resource or efficiencies for introducing variables into the basis in rank order to provide an optimal or good starting feasible solution. Second, it provides an effective method for aggregating constraints of the integer zero-one problem into one constraint to reduce computation times during program execution. Third, it explores an iterative method of "backtracking" to improve some initially computed bound on the optimal solution.

The algorithm is a simple and effective heuristic for solving capital budgeting or project selection type zero-one programming problems. It was tested using a set of thirty-four well known integer programming test problems from the literature and was compared to other known heuristic methods of zero-one problem solving. The algorithm achieved the optimal solution for 50% of the test problems and more than 90% of the optimal solution for most others. Run times are comparable to other heuristic methods studied and for those comparisons made, were faster than a well-known commercial software package.

The algorithm is flexible in terms of potential for additional improvement. Suggestions for further areas of research are provided.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

Without the assistance of many, directly or indirectly, this work would not have been possible.

First, without the U.S. Army's faith in my capabilities to achieve a Masters degree, and its financial support, I would not be here.

Secondly, I wish to commend the Colorado School of Mines for the high level of academic standards it fosters. In particular, I thank Dr. Woolsey and the rest of the Operations Research team, Dr. Maurer and Professor Astle for encouraging a practical hands on approach to mathematical problem solving that instilled in me the confidence to approach established foundations in mathematics from my own perspectives.

Finally, I thank my wife Lydia for her strength, support and encouragement through this challenging period in our lives.

Chapter 1

INTRODUCTION

1.1 Zero-One Programming

Zero-One programming is a subclass of integer programming where variables only take on the values of zero or one to attain a solution. The general form of the zero-one problem is:

$$\text{Maximize } Z = \sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } \sum_{j=1}^{n} a_{i,j} X_j <= b_i \quad i=1,\ldots,m \tag{1.1}$$

$$\text{and } X_j = 0 \text{ or } 1 \text{ for } j= 1,2,3,\ldots,n$$

(or)

$$\text{Minimize } Z = \sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } \sum_{j=1}^{n} a_{i,j} X_j >= b_i \quad i=1,\ldots,m \tag{2.1}$$

$$\text{and } X_j = 0 \text{ or } 1 \text{ for } j= 1,2,3,\ldots,n$$

Since the advent of theory in solving zero-one problems in the mid-1960s most current methods have evolved from a few explicit enumeration [1], [7], [15] or heuristic

methods [5], [6], [10].

Explicit enumeration explores all possible binary solutions in approaching optimality. In some cases, such as the method of Lawler and Bell [7], only those nodes that may improve some bound on the optimal solution (implicit enumeration) are explored. However, these methods usually start at an optimal, but not feasible, point [18].

Two major drawbacks of these methods are the time and computer space involved in exploring all possible binary solutions and according to Gomory [3], dual integer programming codes do not produce a feasible solution until the optimal solution is found. Therefore one often ends up with a good bound that is infeasible. Also, implicit enumeration schemes work well provided the number of variables is less than 130 [18].

Heuristic methods have been developed to provide quick, feasible, and sometimes optimal solutions. Though enumerative methods more often find the optimal solution, heuristic methods are often faster for larger problems [12]. Some examples of heuristic methods are the methods of Senju and Toyoda [10], Hillier [5], and Kochenberger, McCarl and Wyman [6].

The issue often becomes whether one wants a good feasible solution now or the optimal solution later. Further, what is the point of using an optimal code on data that at best will vary 10% in the objective function, the right hand sides, and the matrix coefficients [18]?

1.2 The Approach to the Proposed Algorithm

We employ a heuristic method that effectively computes the efficiencies of each variable based on objective function coefficients, resources consumed by each variable, and total resources available. To make the algorithm more efficient, we aggregate the constraints through a straightforward method explained in Section 3.2.2. No information is lost from the original constraints and this aggregation apparently results in an efficient execution of the algorithm code. Only one constraint must be tested for violation during each iteration.

Explained in Chapter 3, the proposed algorithm uses a method of "backtracking". Backtracking attempts to improve some computed bound on the optimal solution. This however, may substantially increase the number of iterations and processing times for some problems.

From a program user perspective, more quickly computed bound may be a good enough solution. The improvement in the algorithm solution obtained through backtracking may sometimes not be worth the additional processing time necessary.

Chapter 2

CAPITAL BUDGETING MODELS

2.1 General

Woolsey [18] makes the case that the formulation and solution of capital budgeting models is the way to increase utilization of integer programming in the real world. These models involve project selection decisions that must be made under manpower, money, or material constraints. Consider again problem (1.1):

$$\text{Maximize } Z = \sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } \sum_{j=1}^{n} a_{i,j} X_j <= b_i \quad i=1,...,m \qquad (2.1)$$

$$\text{and } X_j = 0 \text{ or } 1 \text{ for } j= 1,2,3,...,n$$

The objective function is the total profit or payoff from doing some collection of the projects X_j, for j=1,...,n. If $X_j = 0$, then project X_j is not selected to maximize the objective function; X_j=1 means project X_j is selected.

Constraint 1, the cost constraint, stipulates that the total cost to carry out the selected projects cannot exceed b_1, the money available to finance operations. Constraint m, the materials constraint, says that the total materials available to carry out the selected projects cannot exceed b_m, the materials available to support operations.

This analogy shows and supports the assertion that zero-one problem solving is ideal for formulating and solving the capital budgeting model. The zero-one algorithm we are searching for should at best find an optimal solution for the above system; alternatively it should find a good feasible solution.

The zero-one additive algorithms of Lawler and Bell [7], Weingartner [15] and Balas [1] or the heuristics of Senju and Toyoda [10], Hillier [5] and Kochenberger, McCarl and Wyman [6] are some well-known methods for solving capital budgeting problems. Each of these methods has particular advantages or drawbacks. The methods of Lawler and Bell, Balas and Senju and Toyoda are discussed in the following sections as detailed examples of enumeration and heuristic methods.

2.2 The Method of Lawler and Bell

The method of Lawler and Bell [7] is a method of solving discrete optimization problems with monotonic non-increasing objective functions. The method makes a conversion of the capital budgeting maximization model as in system (2.1) into a minimization model of the following form:

$$\text{Minimize } Z = \sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } \sum_{j=1}^{n} a_{i,j} X_j >= b_i \quad i=1,\ldots,m \tag{2.2}$$

$$\text{and } X_j = 0 \text{ or } 1 \text{ for } j= 1,2,3,\ldots,n$$

The method uses a series of conditions to exclude some binary solutions and is therefore considered to be a partial enumeration method. It requires only a number of binary additions and subtractions to approach an optimal solution. The practical application of this method is limited to a fairly small number of variables and it has usually not been considered a useful integer programming model since it does not appear to be as effective as the Balas algorithm [18]. Also as indicated above, it requires a conversion of the problem from the maximization to minimization form. The method additionally requires a reordering of constraints and ranking of variables to efficiently find a good feasible or optimal solution [18].

2.3 The Method of Balas

The method of Balas [1] is an additive algorithm that has the advantages of requiring only binary operations and it is free from rounding errors. This method also uses a series of tests to exclude certain branches during a branch and bound process. However, avoiding complete enumeration depends upon the efficiency ranking of variables and during each iteration the constraints must be individually tested for violation. The method also makes a transformation of the capital budgeting maximization model as in system (2.1) into a minimization model of the following form:

$$\text{Minimize } Z = \sum_{j=1}^{n} C_j X_j$$

$$\text{Subject to } \sum_{j=1}^{n} a_{i,j} X_j >= b_i \quad i=1,\ldots,m \tag{2.3}$$

$$\text{and } X_j = 0 \text{ or } 1 \text{ for } j= 1,2,3,\ldots,n$$

Balas' method, without some modifications, makes it necessary to retain most potential solutions within computer storage space and becomes limited for problems with greater than 50 variables according to Petersen [8]. The number of iterations required to solve larger problems often becomes excessive. And again there is the requirement for conversion to the minimization form.

2.4 The method of Senju and Toyoda

The method of Senju and Toyoda [10], [13] is a heuristic that quickly approaches a good feasible or perhaps an optimal solution. Four variations of the heuristic have been explored and tested [10], [13], [16]. This section explains the dual effective gradient method which solves capital budgeting type maximization problems in the form of equations (2.1). It is a relatively simple method that only requires calculation of efficiences for each variable and slacks for each constraint.

The problem starts dual feasible with all variables in the basis. Based on the efficiencies, some variables are removed from the basis until all constraints are satisfied. Any remaining slack is then taken up by some variable that was previously removed

from the basis. This necessitates checking the slack for each constraint while approaching optimality. The above process is done until no additional variables can enter the basis. The key difference between the method of Senju and Toyoda and the proposed method of this thesis , shown in Section 3.6, is the computation of efficiencies.

For example consider a maximization problem in which a variable has only negative matrix coefficients or the columns sum to a negative value. The effectiveness calculation of Senju and Toyoda in some cases penalizes a variable for consuming negative resources giving it a negative or smaller efficiency. This will be demonstrated in Section 3.5, and shows that the proposed method may calculate efficiences in a more appropriate manner for ordered entry of variables into the basis. Also shown is that the proposed algorithm may be a more flexible heuristic method in terms of solving capital budgeting models (1.1), (1.2) or a model with both greater than or equal to, and less than or equal to, constraints. There are also no limitations on the values of the matrix coefficients, $a_{i,j}$'s or objective function coefficients, C_j's.

Chapter 3

THE PROPOSED ALGORITHM

3.1 Background

This chapter will explain in detail the methodology behind the algorithm proposed in this thesis. It is a heuristic method which utilizes payoff-per-unit-of-resource theory for computation of efficiencies. This method is explained in section 3.2.1. A method of constraint aggregation for zero-one problems used in the algorithm is explained in section 3.2.2. Finally a method of "backtracking" to approach the optimal solution (if not already attained) using the above methods is explained in section 3.4.1.

The algorithm provides an optimal solution or a good feasible starting solution for zero-one maximization problems with less than or equal to constraints, and minimization problems with greater than or equal to constraints. In addition, it can be easily tailored to solve maximization or minimization problems with both greater than or equal to and less than or equal to constraints. As previously stated, the algorithm does not require a tedious branch and bound process, but cannot guarantee optimality.

3.2 The Algorithm

The methods of computation of efficiencies and constraint aggregation used in the algorithm are explained in the next two sections.

3.2.1 Efficiency Computation

The algorithm uses a payoff-per-unit-of-combined-resource method similar to the method of Senju and Toyoda [10]. However, we propose here a modification which appears to effectively determine efficiencies for each variable for ordered entry into the basis. These variables are introduced into the algorithm in descending payoff-per-unit -of-combined-resource order for maximization problems and in ascending order for minimization problems. Generally, the variable that has the largest objective function coefficient and consumes the least resources will have the greater efficiency. Computation of efficiences is described below.

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

$$\begin{aligned} \text{Maximize } F= \quad & C_1X_1 + C_2X_2 \\ \text{Subject to} \quad & g_1= a_{1,1}X_1 + a_{1,2}X_2 \ <= b_1 \\ & g_2= a_{2,1}X_1 + a_{2,2}X_2 \ <= b_2 \\ & X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.1}$$

STEP 1: Divide constraints g_1 and g_2 right hand sides by b_1 and b_2 respectively to get:

$$\begin{aligned} \text{Maximize } F= \quad & C_1X_1 + C_2X_2 \\ \text{Subject to} \quad & g_1' <= 1 \\ & g_2' <= 1 \\ & X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.2}$$

STEP 2: Add constraints from the given system of equations (3.2) to get:

$$\begin{aligned} &\text{Maximize } F= \quad C_1X_1 + C_2X_2 \\ &\text{Subject to} \qquad g' <= 2 \\ &\qquad\qquad\quad X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.3}$$

STEP 3: Divide the constraint of (3.3) its by RHS to get:

$$\begin{aligned} &\text{Maximize } F= \quad C_1X_1 + C_2X_2 \\ &\text{Subject to} \qquad g'' <= 1 \\ &\qquad\qquad\quad X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.4}$$

STEP 4: Multiply C(j) by (1-A"(j))

where C(j) is the objective function coefficient for variable X(j), and A"(j) is the constraint coefficient from (3.4) for variable X(j), for all j to get:

E(j)=Efficiency for variable X(j)

NOTE: Step 4 measures the payoff-per-unit-of-combined-resource for each variable.

STEP 5: Introduce variables X(j) into the basis in decreasing order of efficiency for maximization problems and increasing order for minimization problems.

3.2.2 Constraint Aggregation

Shown below, the following method aggregates constraints g_1 and g_2 from system of equations (3.1) into one constraint with the result that when the aggregated constraint is satisfied, so are the original constraints. The reverse also is true. This method has no limitations on the number of constraints and is what provides easy computer coding of the algorithm and faster convergence towards the optimum solution. This method can also be done by hand without great difficulty for smaller problems. The method is as described below.

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

$$\begin{aligned} \text{Maximize } F= \quad & C_1X_1 + C_2X_2 \\ \text{Subject to } g_1= \quad & a_{1,1}X_1 + a_{1,2}X_2 <= b_1 \\ g_2= \quad & a_{2,1}X_1 + a_{2,2}X_2 <= b_2 \\ & X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.5}$$

STEP 1: Divide constraints g_1 and g_2 by right hand sides b_1 and b_2 respectively to get:

$$\begin{aligned} \text{Maximize } F= \quad & C_1X_1 + C_2X_2 \\ \text{Subject to} \quad & g_1' <= 1 \\ & g_2' <= 1 \\ & X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.6}$$

STEP 2: Add constraints from (3.6) to get a one constraint system:

$$\begin{aligned} \text{Maximize } F= \quad & C_1X_1 + C_2X_2 \\ \text{Subject to} \quad & g_1' + g_2' <= 2 \\ & X_1, X_2 = 0 \text{ or } 1 \end{aligned} \tag{3.7}$$

NOTE: Notice the the right hand side of (3.7) is the number of constraints. Normally one could satisfy the above aggregated constraint from (3.7), ie. $g_1'=1.2$ and $g_2'=.8$. However, constraint g_1' from (3.6) is violated though the aggregated constraint is satisfied.

STEP 3: Now raise g_1' and g_2' of (3.7) to some large power W to get:

$$(g_1')^W + (g_2')^W <=2$$

NOTE: Doing this before aggregation gives the same result since the right hand sides are equal to 1.

STEP 4: Now consider as above with $g_1'=1.2$ and $g_2'=.8$.

$$(1.2)^{100} + (.8)^{100} <=2$$

The g_1' term blows up to 82817974.522, and the g_2' term becomes $2.04E^{-10}$. The aggregated constraint is violated as well as the g_1' term. This is our desired result.

The aggregated constraint should be violated when any of the original constraints are.

STEP 5: Now consider when both g_1' and g_2' are less than 1, ie., $g_1'=.5$ and $g_2'=.8$.

$$(.5)^{100} + (.8)^{100} <= 2$$

The aggregated constraint (3.7) is always satisfied and so are the original constraints (3.6) since $g_1'=7.89E^{-31}$, and $g_2'=2.04E^{-10}$.

NOTE: In terms of the sensitivity of the above method to changes in the exponent W, $W=10^8$, would allow no constraint to be larger than 1.00000008 in the above example. A similar process works for minimization problems except that g_1' and g_2' are raised to the power, -W. See Appendix H for further discussion on choosing W.

3.2.3 Backtracking

Backtracking is a method used in some heuristic algorithms to improve a computed bound on the optimal solution through some iterative scheme.

As stated earlier, the proposed algorithm is a heuristic method which ultimately means it does not guarantee an optimal solution. Some problems that the algorithm does not optimize are given in Chapter 4. Those problems can be generally characterized as having substantial differences in numerical value between the largest objective function coefficients and the smallest. Therefore, a heuristic that uses efficiencies to approach the optimal solution will sometimes not be optimal if, given some computed lower bound on the optimal solution based on efficiencies, the following conditions hold. First, the variables with smaller objective function coefficients sum to a larger value than that of

some single variable in the basis. The second condition is that if these variables were together in the basis instead of the single variable, they also do not violate the aggregated constraint. These conditions occur in some of the test problems that the algorithm does not optimize. Backtracking provides the optimal solutions for some problems. The allocation problem results (Section 4.4) provide good examples where the algorithm optimizes some problems and does not optimize others when changes in are made in the amount of resources available. This change in resources available allows some combination of variables with smaller objective function coefficients and a combined greater payoff to be in the basis instead of some variable with a singly large payoff. Examples are allocation problems 3, 5, 8, from [14].

I would argue that the above conditions will hold only in some unusual cases of the capital budgeting model. For example, in most cases I would say that business organizations would choose to do one or two projects rather than twenty or thirty if the profit from doing the latter marginally exceeds the former.

In any case, the proposed method of backtracking attempts to approach the optimal solution as closely as possible while keeping processing time at an acceptable level. The following steps explain how this backtracking is accomplished.

STEP 1: The computation of efficiencies is as in the algorithm,

$E(j) = C(j)*(1- A''(j))$ for all $j=1,2,...,n$. The variable subscripts are then put into a rank order array, PSN(X) based on efficiencies, ie., for a maximization problem.

PSN(X)= 7 6 10 2 5 8 4 3 1 9

STEP 2: Initially the variables are entered into the basis from left to right, until the aggregated constraint is no longer satisfied, to get a lower bound on the optimal solution.

STEP 3: The variables are then taken out of the basis from right to left, one at a time, and another forward iteration is done to try to improve the lower bound on the optimal value.

STEP 4: During this process, any feasible solution found which is larger than the previous becomes the new lower bound. The largest value achieved is the algorithm solution.

This process increases run times but provides the optimal solution in many cases. Generally, those problems that the algorithm does not optimize have one variable with a large objective funtion coefficient, where some combination of variables with smaller objective function coefficients could instead be in the basis, and together contribute a greater payoff. The pattern for which the algorithm does not optimize is pretty clear, and in some cases, can be seen by just noting the range of the objective function coefficients.

3.3 Flowchart Discussion

Figures 1, 2 and 3 outline the coded algorithm. Figure 1 explains the initial computation of efficiencies followed by a branch to the maximization process (Figure2) or the minimization process (Figure 3). The variable T, in the flowcharts, represents the aggregated constraint which results from the procedure outlined in Section 3.2.2.

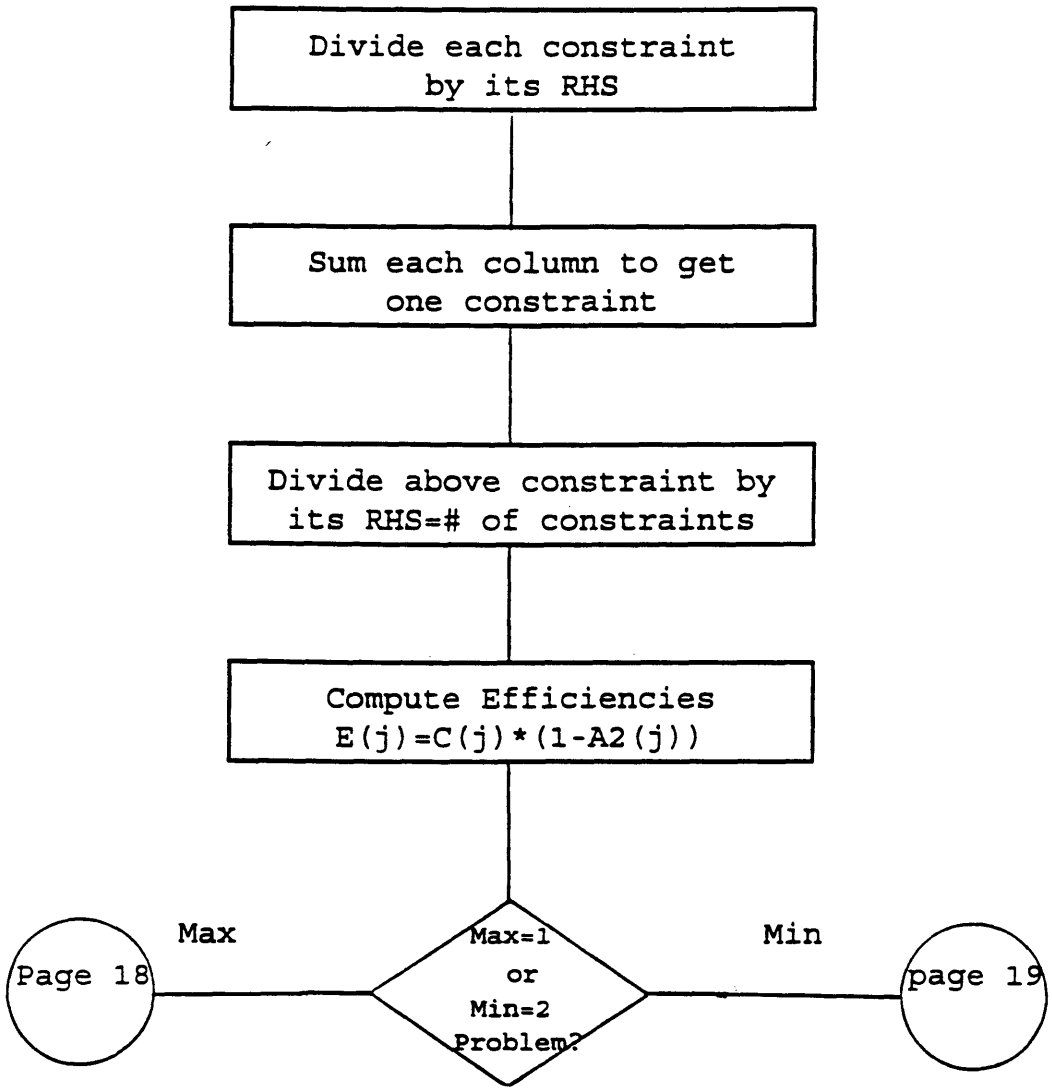

FIGURE 1

THE ALGORITHM FLOWCHART

NOTE: A2(j) is the sum of the constraint coefficients for variable j divided by the number of constraints.

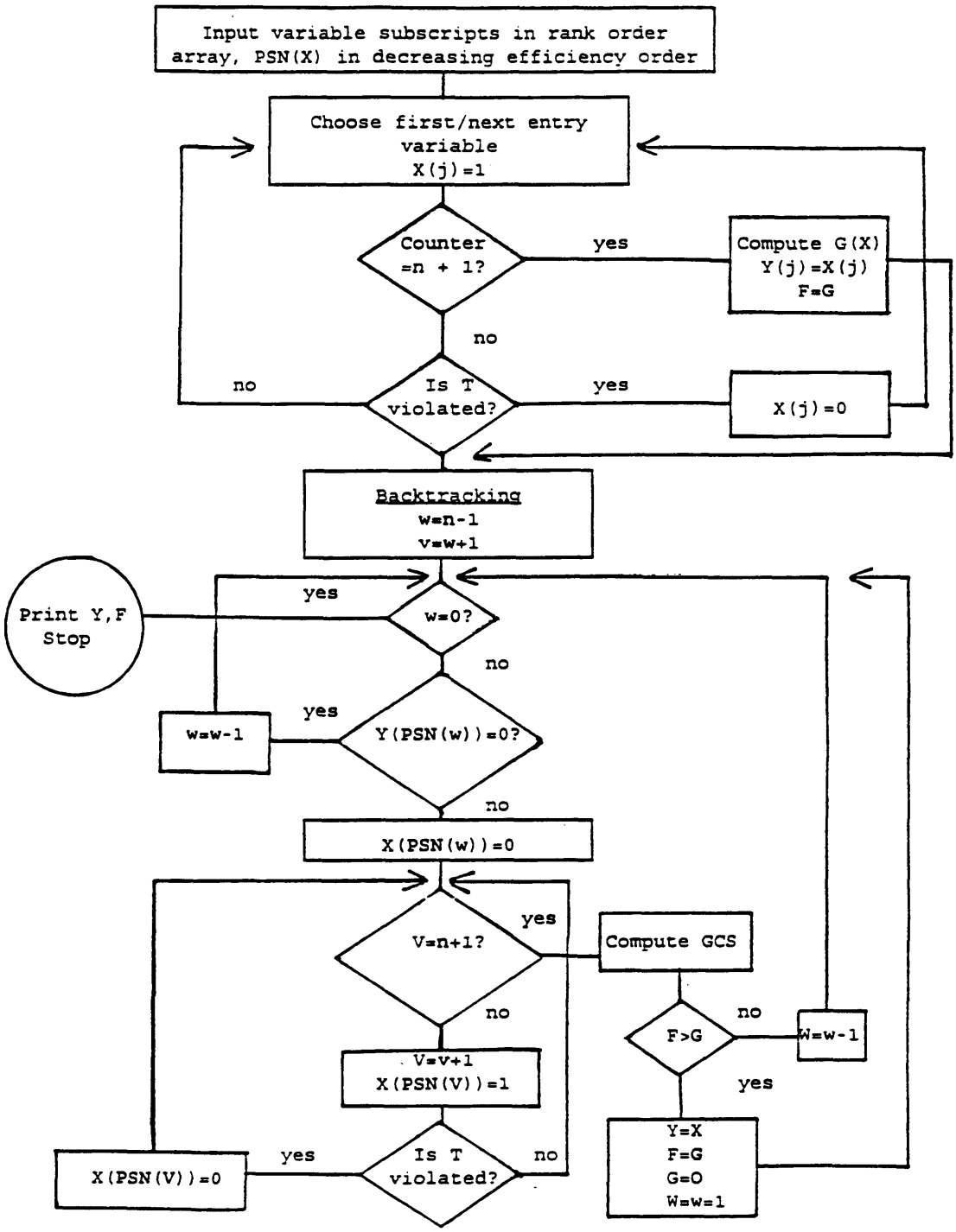

FIGURE 2

MAXIMIZATION BRANCH

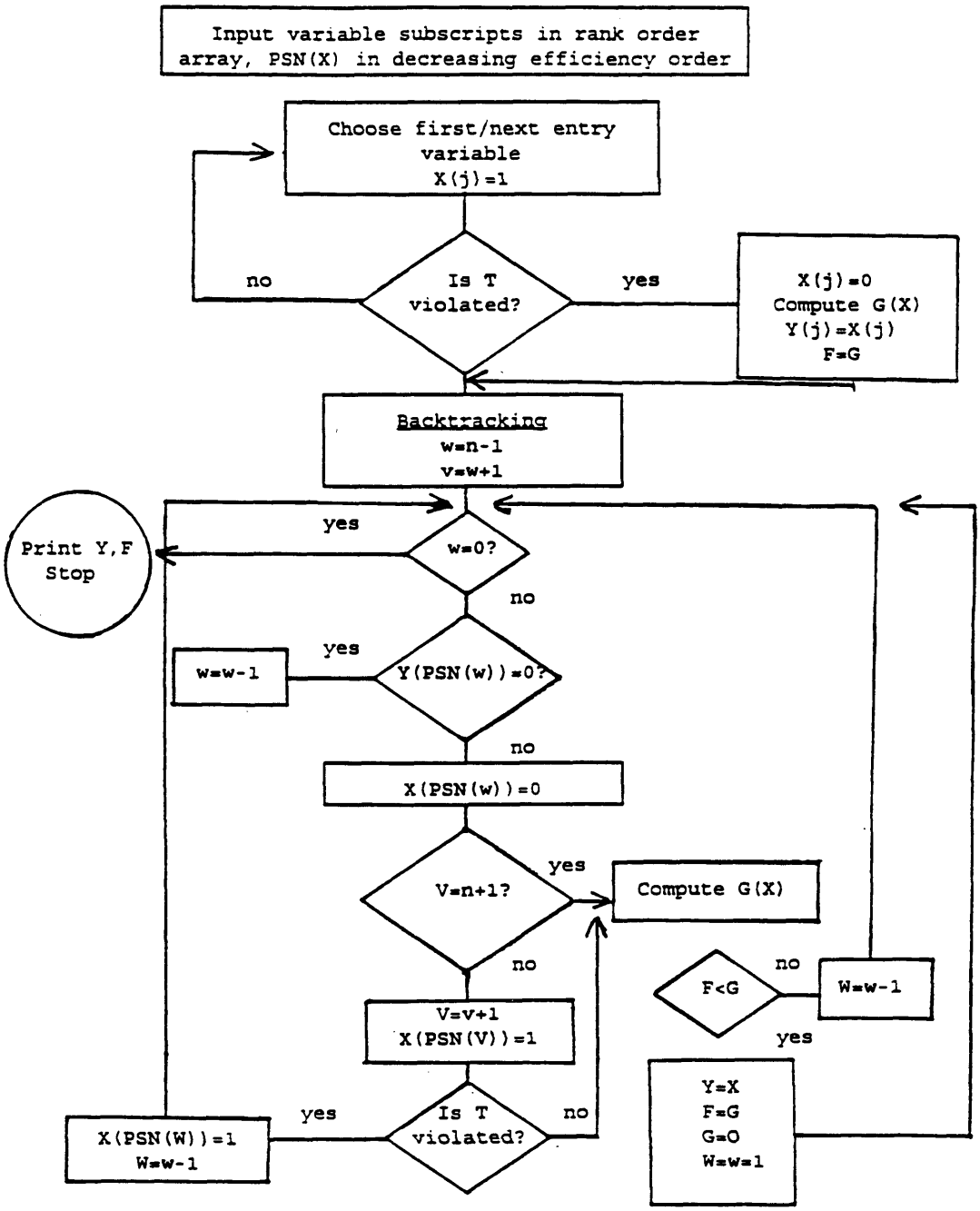

FIGURE 3

MINIMIZATION BRANCH

3.4 Example Problems

The following sections illustrate the use of the proposed algorithm to solve a few simple problems. Shown are a maximization example problem of the form of equations (1.1), a minimization example problem of the form of equations (1.2), and a maximization example with both less than or equal to and greater than or equal to constraints. Not shown is the method of constraint aggregation since it is only used in the coded algorithm, and is not necessary for these simple illustrations.

3.4.1 A Maximization Problem Example

This section illustrates the algorithm using a simple four variable maximization problem. Results are calculated manually.

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

$$\begin{aligned} \text{Maximize } F &= 4X_1 + 3X_2 + 2X_3 + 1X_4 \\ \text{Subject to } g_1 &= 7X_1 + 2X_2 + 1X_3 + 4X_4 <= 7 \\ g_2 &= 2X_1 + 3X_2 + 1X_3 + 4X_4 <= 5 \\ X_1, X_2, X_3, X_4 &= 0 \text{ or } 1 \end{aligned} \tag{3.8}$$

STEP 1: Divide constraints g_1 and g_2 by right hand sides 7 and 5 respectively to get:

$$\begin{aligned} \text{Maximize } F &= 4X_1 + 3X_2 + 2X_3 + 1X_4 \\ \text{Subject to } g_1' &= 1X_1 + (2/7)X_2 + (1/7)X_3 + (4/7)X_4 <= 1 \\ g_2' &= (2/5)X_1 + (3/5)X_2 + (1/5)X_3 + (4/5)X_4 <= 1 \end{aligned} \tag{3.9}$$

$$X_1, X_2, X_3, X_4 = 0 \text{ or } 1$$

STEP 2: Add constraints from system of equations (3.9) to get:

$$\begin{aligned} &\text{Maximize } F= \ 4X_1 + 3X_2 + 2X_3 + 1X_4 \\ &\text{Subject to } \ (7/5)X_1 + (31/35)X_2 + (12/35)X_3 + (48/35)X_4 <= 2 \\ &\qquad X_1, X_2, X_3, X_4 = 0 \text{ or } 1 \end{aligned} \tag{3.10}$$

STEP 3: Divide constraint of (3.10) by its RHS to get:

$$\begin{aligned} &\text{Maximize } F= \ 4X_1 + 3X_2 + 2X_3 + 1X_4 \\ &\text{Subject to } \ (7/10)X_1 + (31/70)X_2 + (12/70)X_3 + (48/70)X_4 <= 1 \\ &\qquad X_1, X_2, X_3, X_4 = 0 \text{ or } 1 \end{aligned} \tag{3.11}$$

STEP 4: Multiply C(j) by (1-A"(j))

where C(j) is the objective function coefficient for variable X(j), and A"(j) is the constraint coefficient from (3.11) for variable X(j), for all j to get Efficiency, E(j):

E(1) = 4(1-7/10) = 6/5

E(2) = 3(1-31/70) = 117/70

E(3) = 2(1-12/70) = 58/35

E(4) = 1(1-11/35) = 11/35

rank order array PSN(X) = (2,3,1,4)

STEP 5: Introduce variables X(j) into the basis in decreasing order of efficiency for this maximization problem gives the following optimal result:

X_1	X_2	X_3	X_4	Z
0	1	1	0	5

STEP 6: Backtracking to achieve a possibly better result requires two additional iterations setting first X_3=0,step 6A, and then X_2=0,step 6B, to get the following:

STEP 6A: Setting X_3= 0 and iterating gives

X_1	X_2	X_3	X_4	Z
0	1	0	0	3

which does not improve our solution.

STEP 6B: Setting X_2= 0 and iterating gives

X_1	X_2	X_3	X_4	Z
0	0	1	1	3

which also does not improve our solution.

Given the above backtracking, we now know that our solution from Step 5 is the best that the algorithm will give.

3.4.2 A Minimization Problem Example

This section illustrates the algorithm using a simple three variable minimization problem. Results are calculated manually.

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

$$\begin{aligned} &\text{Minimize } F = 4X_1 + 5X_2 + 3X_3 \\ &\text{Subject to } g_1 = 1X_1 + 1X_2 + 2X_3 >= 3 \\ &\qquad g_2 = 1X_1 + 2X_2 + 1X_3 >= 3 \\ &\qquad X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.12}$$

STEP 1: Divide constraints g_1 and g_2 by right hand sides 3 and 3 respectively to get:

$$\begin{aligned} &\text{Minimize } F = 4X_1 + 5X_2 + 3X_3 \\ &\text{Subject to } g_1' = (1/3)X_1 + (1/3)X_2 + (2/3)X_3 >= 1 \\ &\qquad g_2' = (1/3)X_1 + (2/3)X_2 + (1/3)X_3 >= 1 \\ &\qquad X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.13}$$

STEP 2: Add constraints from system of equations (3.13) to get:

$$\begin{aligned} &\text{Minimize } F = 4X_1 + 5X_2 + 3X_3 \\ &\text{Subject to } (2/3)X_1 + 1X_2 + 1X_3 >= 2 \\ &\qquad X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.14}$$

STEP 3: Divide constraint of (3.14) its by RHS to get:

$$\begin{aligned} &\text{Minimize } F= \ 4X_1 + 5X_2 + 3X_3 \\ &\text{Subject to} \quad 2/6X_1 + 1/2X_2 + 1/2X_3 \ >= 1 \\ &\qquad X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.15}$$

STEP 4: Multiply C(j) by (1-A"(j))

where C(j) is the objective function coefficient for variable X(j), and A"(j) is the constraint coefficient from (3.15) for variable X(j), for all j to get Efficiency, E(j):

E(1) = 4(1-2/6) = 8/3

E(2) = 5(1-1/2) = 5/2

E(3) = 3(1-1/2) = 3/2

rank order array PSN(X) = (3,2,1)

STEP 5: Introducing variables X(j) into the basis in increasing order of efficiency for this minimization problem gives the following optimal result:

X_1	X_2	X_3	Z
0	1	1	8

STEP 6: Backtracking to achieve a possibly better result requires two additional iterations setting first X_2=0, step 6A, and then X_3=0, step 6B to get the following:

STEP 6A: Setting X_2= 0 and iterating gives

X_1	X_2	X_3	Z
1	0	1	7

which however, is infeasible. Constraint 2 is not satisfied.

STEP 6B: Setting $X_3 = 0$ and iterating gives

X_1	X_2	X_3	Z
1	1	0	9

which also is infeasible. Constraint 1 is not satisfied.

Given the above backtracking, we now know that our solution from Step 5 is the best that the algorithm will give.

3.4.2 A Maximization Problem Example With Less Than and Greater Than Constraints

This section illustrates the algorithm using a simple three variable maximization problem. This example serves to illustrate that the algorithm could be coded to solve problems with both less than or equal to and greater than or equal to constraints. Results are calculated manually.

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

$$\begin{aligned} \text{Maximize } F= \quad & 4X_1 + 5X_2 + 3X_3 \\ \text{Subject to } \quad & g_1= 1X_1 + 1X_2 + 2X_3 \ <= 3 \\ & g_2= 1X_1 + 2X_2 + 2X_3 \ >= 2 \\ & X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.16}$$

STEP 1: Divide constraints g_1 and g_2 by right hand sides 3 and 2 respectively to get:

$$\begin{aligned} \text{Maximize } F= \quad & 4X_1 + 5X_2 + 3X_3 \\ \text{Subject to } \quad & g_1'= (1/3)X_1 + (1/3)X_2 + (2/3)X_3 <= 1 \\ & g_2'= (1/2)X_1 + 1X_2 + 1X_3 \ >= 1 \\ & X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.17}$$

STEP 2: Add constraints from the system of equations of (3.17) to get:

$$\begin{aligned} \text{Maximize } F= \quad & 4X_1 + 5X_2 + 3X_3 \\ \text{Subject to } \quad & (5/6)X_1 + (4/3)X_2 + (5/3)X_3 \cong 2 \\ & X_1, X_2, X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.18}$$

STEP 3: Divide constraint of (3.18) its by RHS to get:

$$\begin{aligned} \text{Maximize } F= \quad & 4X_1 + 5X_2 + 3X_3 \\ \text{Subject to } \quad & 5/12X_1 + 4/6X_2 + 5/6X_3 \cong 1 \\ & X_1,X_2,X_3 = 0 \text{ or } 1 \end{aligned} \tag{3.19}$$

STEP 4: Multiply C(j) by (1-A"(j))

where C(j) is the objective function coefficient for variable X(j), and A"(j) is the constraint coefficient from (3.19) for variable X(j), for all j to get Efficiency, E(j):

E(1) = 4(1-5/12) = 7/3

E(2) = 5(1-4/6) = 5/3

E(3) = 3(1-5/6) = 1/2

rank order array PSN(X) = (1,2,3)

STEP 5: Introducing variables X(j) into the basis in increasing order of efficiency for maximization problems until both constraints g_1, and g_2 are satisfied gives the following optimal result:

X_1	X_2	X_3	Z
1	1	0	9

g1=1(1) + 1(1) + 1(0) <=3 and

g2=1(1) + 2(1) + 1(0) >=2 so both constraints are satisfied.

STEP 6: Backtracking to achieve a possibly better result requires two additional iterations setting first X_2=0, Step 6A, and then X_1=0, Step 6B, to get the following:

STEP 6A: Setting X_2= 0 and iterating gives

X_1	X_2	X_3	Z
1	0	1	7

which is feasible but does not improve Z.

STEP 6B: Setting $X_1 = 0$ and iterating gives

X_1	X_2	X_3	Z
0	1	1	8

which also is feasible, but does not improve Z.

Given the above backtracking, we now know that our solution from Step 5 is the best that the algorithm will give.

3.5 Two Comparison Examples With Senju and Toyoda

The following sections provide the results of two examples comparing the Senju and Toyoda Method [10] and the proposed algorithm.

3.5.1 Example 1

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

Maximize $F = 500X_1 + 800X_2 + 300X_3 + 600X_4 + 100X_5 + 300X_6$

Subject to $1X_1 + 4X_2 + 5X_3 + 7X_4 + 5X_5 + 6X_6 <= 16$

$5X_1 + 3X_2 + 2X_3 + 8X_4 + 4X_5 + 9X_6 <= 16$

$X_1, X_2, X_3, X_4, X_5, X_6 = 0$ or 1

With Optimal Solution:

X_1	X_2	X_3	X_4	X_5	X_6	Z
1	1	0	1	0	0	1900

Senju and Toyoda results are:

Variable	X_1	X_2	X_3	X_4	X_5	X_6	Z
Efficiencies	5.7	9.7	3.3	2.9	0.8	1.4	-
Rank Order	2	1	3	4	6	5	-
Solution	1	1	1	0	1	0	1700

The proposed algorithm results are:

Variable	X_1	X_2	X_3	X_4	X_5	X_6	Z
Efficiencies	406.3	625	234.4	318.8	71.88	159.4	-
Rank Order	2	1	4	3	6	5	-
Solution	1	1	0	1	0	0	1900

3.52 Example 2

I asserted in Section 2.4 that proposed algorithm may more appropriately calculate efficiences than the method of Senju and Toyoda. The following example supports that assertion.

GIVEN THE FOLLOWING SYSTEM OF EQUATIONS:

$$\text{Maximize } F = 4X_1 + 10X_2 + 8X_3 + 12X_4$$

$$\text{Subject to } 1X_1 - 2X_2 + 1X_3 + 5X_4 <= 6$$

$$2X_1 - 1X_2 + 4X_3 + 3X_4 <= 4$$

$$X_1, X_2, X_3, X_4 = 0 \text{ or } 1$$

With Optimal Solution :

X_1	X_2	X_3	X_4	Z
1	1	0	1	26

Senju and Toyoda results are :

Variable	X_1	X_2	X_3	X_4	Z
Efficiencies	57	-5	.53	1.7	-
Rank order	2	4	3	1	-
Solution	1	1	0	1	26

The proposed algorithm results are:

Variable	X_1	X_2	X_3	X_4	Z
Efficiencies	2.7	12.9	3.3	2.5	-
Rank order	3	1	2	4	-
Solution	1	1	0	1	26

Though both methods achieve the optimum result, analysis of the computed efficiencies should show that the proposed method does a more appropriate calculation as asserted. For example, consider the efficiencies computed by both methods for the variable X_2. The variable X_2 has the second largest objective function coefficient, and is the only variable to use negative resources. One would expect that its effectiveness calculation would be among the highest. The Senju and Toyoda calculation for E(2) is -5, which is smallest. The proposed method calculates E(2) as 12.9, which is largest. That result is consistent with the results of all problems considered. Based on the computational experience of this thesis, we

assert that this algorithm gives a solution equal to or better that that of the Method of Senju and Toyoda. See Section 4.2 for other comparison test problems.

Chapter 4
COMPUTATIONAL RESULTS AND DISCUSSION

4.1 General

This chapter will summarize the results of this study, and reiterate those problem areas and successes found using this algorithm. Also contained are computational results and/or comparisons using thirty-four well known test problems from the literature. These test problems include capital budgeting, allocation and knapsack problems.

The allocation problems are of the same form as the capital budgeting model explained earlier [14]. They are used to investigate the sensitivity of the proposed algorithm to a relatively small change in resource availability.

Knapsack problems are also well known and of the same form as the capital budgeting model. The name "knapsack" refers to maximizing the value of items carried by a hiker, for instance, while constrained to a certain weight and/or volume limitation. Applications of this model range from cargo-loading to project selection and cutting stock problems [12].

Section 4.2 gives algorithm results of four problems used to test the Senju and Toyoda heuristic from [13]. Solutions found using the Senju and Toyoda heuristic are listed for comparison.

Section 4.3 makes a comparison of run times and iterations or nodes considered for the proposed algorithm and a commercial software package, Storm [11]. Problems used are three general capital budgeting problems from [9], and nine allocation problems from [14].

Section 4.4 contains algorithm results for Petersen's test problem set [8]. Optimal solutions are given for comparison with the proposed algorithm results.

Section 4.5 provides algorithm results for a well known set of nine knapsack problems from [15]. Again, optimal solutions are given for comparison with the proposed algorithm results.

Finally, section 4.6 tests the proposed algorithm in solving a set of Haldi's [4] minimization test problems.

Algorithm problem run times are on a Zenith 386-SX personal computer. An iteration is defined as the introduction of a variable into the basis, and testing if the aggregated constraint is satisfied.

4.2 Senju and Toyoda Test Problem Comparisons

This section compares results of four capital budgeting problems [13], using the Senju and Toyoda heuristic and the proposed algorithm. Solutions, computation times and iterations for the Senju and Toyoda method are using a coded version of the Senju and Toyoda dual effective gradient algorithm. Both algorithms were both run on a 386SX personal computer. Run times for the algorithms are given in minutes:seconds. Solutions for the two methods are relatively similar. However, the proposed algorithm run times for

problems 1 and 4 appear excessive when compared with the times for the Senju and Toyoda method. This is due to the backtracking used in the proposed algorithm to attain the best possible algorithm solution. Problems are listed in Appendix A.

TABLE 1

SENJU AND TOYODA TEST PROBLEM RESULTS

Prob	S&T Time	S&T Iterations	Best Obj Func Value	Alg Time	Alg Iterations	Best Obj Func Value
1	01:23	34	8698	03:36	1311	8681
2	00:02	3	1800	00:02	17	1800
3	00:02	3	2600	00:02	32	2600
4	01:39	47	7605	03:21	876	7675

4.3 General Test Problem Results

This section makes a comparison of run times and solutions for the proposed algorithm and a commercial software package, Storm [11] which uses a branch and bound process to achieve the optimal solution. Tested were three capital budgeting problems from [9], and nine allocation problems from [14]. The Storm solution is optimal for all problems. Results indicate that the proposed algorithm run times are roughly one-quarter of Storm run times, at the expense of a not attaining optimality. The algorithm solution was optimal in eight of twelve problems. Problems are listed in Appendices B and E respectively.

TABLE 2

CAPITAL BUDGETING PROBLEM RESULTS

Problem	Storm Solution	Storm Nodes	Storm Run Time	Alg Solution	Alg Iterations	Alg Run Time
1	766	6	00:10	766	25	00:02
2	2033	14	00:10	2033	26	00:02
3	3724	10	00:12	3724	29	00:03

TABLE 3

ALLOCATION PROBLEM RESULTS

Problem	Storm Solution	Storm Nodes	Storm Run Time	Alg Solution	Alg Iterations	Alg Run Time
1	50	8	00:09	50	34	00:02
2	52	16	00:09	52	37	00:02
3	57	14	00:10	53	36	00:02
4	62	8	00:09	62	38	00:02
5	67	14	00:10	61	34	00:02
6	68	9	00:09	67	37	00:02
7	70	9	00:09	70	40	00:02
8	75	13	00:10	72	41	00:02
9	85	14	00:10	85	46	00:02

4.4 Petersen Test Problem Results

This section presents algorithm results of seven well known test problems used by Petersen [8] to evaluate the Balas Algorithm. The results of the Petersen Study (Table 4) are the best results using any of four modifications of the original Balas algorithm [1]. The proposed algorithm achieved the optimal solution for four of seven test problems and better than 93% of the optimal solution for all problems. In addition, run times and iterations are reasonable when compared with the results of the Petersen study. Problems are listed in Appendix C.

TABLE 4

PETERSEN TEST PROBLEM RESULTS

Prob	Petersen Time	Petersen Iterations	Best Obj Func Value	Alg Time	Alg Iterations	Best Obj Func Value
1	<01:00	5	3800	00:02	16	3800
2	<01:00	9	8706.1	00:04	32	8706.1
3	01:00	63	4015	00:07	92	4015
4	04:00	1805	6120	00:11	122	6120
5	03:00	1041	12400	00:18	274	12200
6	05:00	2628	10618	00:17	278	10002
7	36:00	16778	16537	00:27	449	15508

4.5 Knapsack Test Problem Results

This section provides algorithm results for Weingartner and Ness' [15] multi-dimensional zero-one knapsack problems. Optimality was attained for two of nine problems and over 96% of the optimal solution for all others. Run times and iterations necessary may be excessive for problems 7 and 8. Problems are listed in Appendix E.

TABLE 5

KNAPSACK PROBLEM RESULTS

Prob	Size (NxM)	Optimal Solution	Algorithm Solution	Percent of Optimality	Iterations	Run time
1	28x2	141278	140778	99.6	235	00:08
2	28x2	130883	130883	100	243	00:08
3	28x2	95677	95677	100	161	00:08
4	28x2	119337	115406	96.7	269	00:08
5	28x2	98796	95787	96.9	161	00:08
6	28x2	130623	130233	99.7	239	00:08
7	28x2	1095445	1090201	99.5	5102	01:16
8	28x2	624319	609522	97.6	1845	01:28

4.6 Haldi Test Problem Results

This section tests the proposed algorithm in solving a set of Haldi[4] minimization test problems. The number of iterations and run times were minimal in reaching good feasible solutions. Backtracking for these problems considers whether some variable

can be removed from the basis without violating any constraints. This is because removing one variable from the solution and entering others can only give the same solution or a worse solution, when all the objective function coefficients are the same. Problems are listed in Appendix F.

TABLE 6

HALDI PROBLEM RESULTS

Prob	Size (NxM)	Optimal Solution	Algorithm Solution	Percent of Optimality	Iterations	Run time
1	15x35	9	9	100	99	00:07
2	15x15	10	13	70	117	00:06
3	31x31	18	20	88.9	430	00:22

Chapter 5

CONCLUSIONS AND SUGGESTIONS FOR FURTHER STUDY

5.1 Conclusion

The proposed algorithm provides optimal or good feasible staring solutions for both minimization problems and maximization problems of the given forms (1.1 and 1.2). It provides a valid method of computing variable efficiences for ordered entry into the basis which could possibly make other known algorithms more efficient. The method of constraint aggregation explored appears to be effective and reduces computer processing times for some problems . The algorithm solves a variety of zero-one problems and requires no conversion of problems from maximization to minimization form or vice versa. The optimal solution was attained for approximately fifty percent of the well-known test problems used to evaluate the algorithm. For only two problems did the algorithm not acheive more than ninety percent of the optimal solution. Those were minimization problems which were reasonably close to optimum.

One of the major drawbacks of this method is excessive processing times for some large problems (Sections 4.4 and 4.5). This is due to the method of bactracking used to obtain the best possible solution. This backtracking may not be warranted since it sometimes increases processing time dramatically while only producing a slightly better solution.

Examples are Petersen test problems 5,6 and 7 [8]. Without backtracking, the algorithm gives solutions of 12006, 9520 and 14102 in 18, 15 and 30 seconds respectively.

The algorithm also appears to be sensitive to changes in resource availability indicated by the results of the allocation problems in Section 4.3.

Thirdly, the computer code exhibits overflow problems when the value of the exponent W#, used in constraint aggregation is chosen too large. For the thirty-four test problems used, W# equal to 1000, and W# equal to -100 worked without problems for maximization and minimization problems respectively. As stated earlier, the value of W# chosen can be based on the desired tolerance in allowing a constraint to be violated (See Appendix H). For instance, in a maximization problem with two normalized constraints, $W\#=10^8$, would allow no single constraint left hand side to be larger than 1.00000008. Also at issue is the computer hardware capacity for storing extremely large numbers when constraints are violated and "blow up" causing overflow error. If desired, for a particular problem, W# can be chosen large enough that it causes overflow error and then backed down until it no longer does. This provides the best assurance that no constraints are violated. Alternatively, as in the program listing (Appendix G), an error handling routine can be coded within the program. The listed routine sets T# = m+1, when overflow occurs. This indicates that some constraint is violated. With this error handling routine, W can be made as large as desired, and feasibility is guaranteed.

5.2 Areas for Further Research

We have shown in Section 3.4.3 that the algorithm could also be defined to solve problems with both greater than or equal to, and less than or equal to constraints. This could be an area for further study. None of the other algorithms studied for this thesis have that flexibility.

Another area for further study would be to determine an efficient method to achieve optimality once a bound on the optimal solution has been obtained. As stated, this is the major disadvantage of the proposed algorithm.

Also, the method of choosing W (Appendix H), based on a desired tolerance is suggested as an area of further study.

5.3 Summary

The purpose of this thesis was to establish and test an efficient method for solving integer zero-one programming problems. In particular, we focused on the capital budgeting problem. The algorithm was tested against a number of difficult, well-known test problems. These included test problems of Petersen [8], Weingartner and Ness [15], Haldi [4], Senju and Toyoda [13] and Trauth and Woolsey [14]. Computer processing times were also compared to a commercial software package. In both cases, initial results indicate that the performance of the algorithm was reasonably successful in terms of closeness to optimality and run times. Results obtained were consistent with the results of other programming algorithms studied, and processing times for those problems considered were better than

those of "Storm"[11], though the solutions were not always optimal. The algorithm appears to be a flexible, and viable heuristic with potential for further application.

REFERENCES CITED

1. Balas, E.,1965. "An Additive Algorithm for Solving Linear Programs with Variables". Operations Research, Vol 13, pp517-546.

2. Balas, E. and C. Martin,1980. "Pivot and Complement, A Heuristic for 0-1 Programming". Management Science, Vol. 26, pp86-96.

3. Gomory, R.,1963. "An Algorithm for Integer Solutions to Linear Programming". Recent Advances in Mathematical Programming. McGraw-Hill, New York.

4. Haldi, J.,1964. "Integer Programming Test Problems". Working Paper No. 43". Graduate School of Business. Stanford University.

5. Hillier, F.,1969. "Efficient Heuristic Procedures for Integer Linear Programming with an Interior". Operations Research Vol. 17,pp600-637.

6. Kochenberger, G.A., McCarl, B.A., and Wyman, F.P.,1974. "A Heuristic for General Integer Programming". Decision Sciences, Vol. 5, No. 1, pp36-44.

7. Lawler, E. and M.D. Bell, 1966. "A Method for Solving Discrete Optimization Problems". Operations Research, Vol. 14, pp1098-1112.

8. Petersen, C.,1967. "Computational Experience with Variants of the Balas Algorithm Applied to Selection of R & D Projects". Management Science, Vol. 13, pp736-750.

9. Roussos, G., 1995. "An Application of Multiple Choice Programming to Capital Budgeting Problems". Masters Thesis, Colorado School of Mines, Golden, Co.

10. Senju, S. and Y. Toyoda, 1968. " An Approach to Linear Programming with 0-1 Variables". Management Science, Vol. 15, pp B196-207.

11. "Storm", 1992. Version 2.0. Storm Software, Inc. Cleveland, Ohio.

12. Taha, H., 1982. "Operations Research". McMillan Publishing Co, New York.

13. Toyoda, Y., 1975. "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems". Management Science, Vol. 15, pp481-493.

14.Trauth, C.A. and R.E.D. Woolsey, 1969. "Integer Linear Programming: A Study in Computational Efficiency". Management Science Vol.15, pp481-493.

15. Weingartner, H.M. and D.N. Ness, 1967. "Methods for the Solution of Multi-Dimensional 0/1 Knapsack Problems". Operations Research, Vol. 15, pp83-103.

16. Zanakis, S., 1977. "Heuristic 0-1 Linear Programming: An Experimental Comparison of Three Methods". Management Science, Vol. 24,,pp91-104.

17. Zugail, A.,1991. "A Heuristic for Solving Economic Models Formulated as 0-1 Integer Programming Problems". PhD Thesis, Colorado School of Mines, Golden Co.

18. Woolsey, R.E.D.,1995. Integer Programming Class Notes, Colorado School of Mines, Golden, Co.

Appendix A

Senju and Toyoda Test Problems

Problem 1 [13]

Variables 60,Constraints 30

Maximize Objective Function:

2,77,6,67,930,3,6,270,33,13
110,21,56,974,47,734,238,75,200,51
47,63,7,6,468,72,95,82,91,83
27,13,6,76,55,72,300,6,65,393
63,61,52,85,29,640,558,53,47,25
3,6,568,6,2,780,69,31,774,22

Subject to Constraint Matrix:

47,774,76,56,59,22,42,1,21,760
818,62,42,36,785,29,662,49,608,116
834,57,42,39,994,690,27,524,23,96
667,490,805,46,19,26,97,71,699,465
53,26,123,20,25,450,22,979,75,96
27,41,21,81,15,76,97,646,898,37

73,67,27,99,35,794,53,378,234,32
792,97,64,19,435,712,837,22,504,332
13,65,86,29,894,266,75,16,86,91
67,445,118,73,97,370,88,85,165,268
758,21,255,81,5,774,39,377,18,370
96,61,57,23,13,164,908,834,960,87

36,42,56,96,438,49,57,16,978,9
644,584,82,550,283,340,596,788,33,350
55,59,348,66,468,983,6,33,42,96

464,175,33,97,15,22,9,554,358,587
71,23,931,931,94,798,73,873,22,39
71,864,59,82,16,444,37,475,65,5

47,114,26,668,82,43,55,55,56,27
716,7,77,26,950,320,350,95,714,789
430,97,590,32,69,264,19,51,97,33
571,388,602,140,15,85,42,66,778,936
61,23,449,973,828,33,53,297,75,3
54,27,918,11,620,13,28,80,79,3

61,720,7,31,22,82,688,19,82,654
809,99,81,97,830,826,775,72,9,719
740,860,72,30,82,112,66,638,150,13
586,590,519,2,320,13,964,754,70,241
72,12,996,868,36,91,79,221,49,960
23,18,748,408,688,97,85,777,294,17

698,53,290,3,62,37,704,810,42,17
983,11,45,56,234,389,712,664,59,15
22,91,57,784,75,719,294,978,75,86
105,227,760,2,190,3,71,32,210,678
41,93,47,581,37,977,62,503,32,85
31,36,30,328,74,31,56,891,62,97

71,37,978,93,9,23,47,71,744,9
619,32,214,31,796,103,593,16,468,700
884,67,36,3,93,71,734,504,81,53
509,114,293,31,75,59,99,11,67,306
96,218,845,303,3,319,86,724,22,838
82,5,330,58,55,66,53,916,89,56

33,27,13,57,6,87,21,12,15,290
206,420,32,880,854,417,770,4,12,952
604,13,96,910,34,460,76,16,140,100
876,622,559,39,640,59,6,244,232,513
644,7,813,624,990,274,808,372,2,694
804,39,5,644,914,484,1,8,43,92

16,36,538,210,844,520,33,73,100,284
650,85,894,2,206,637,324,318,7,556

46,818,92,65,520,721,90,53,174,43
320,812,382,16,878,678,29,92,755,827
27,218,143,12,57,480,154,944,7,730
12,65,67,39,390,32,39,318,47,86

45,51,59,21,53,43,25,7,42,27
310,45,72,53,798,304,354,79,45,44
52,76,45,26,27,968,86,16,62,85
790,208,390,36,62,83,93,16,574,150
99,7,920,860,12,404,31,560,37,32
9,62,7,43,17,77,73,368,66,82

11,51,97,26,83,426,92,39,66,2
23,93,85,660,85,774,77,77,927,868
7,554,760,104,48,202,45,75,51,55
716,752,37,95,267,91,5,956,444,529
96,99,17,99,62,7,394,580,604,89
678,476,97,234,1,608,19,69,67,51

410,89,414,81,130,491,6,238,79,43
5,288,910,204,948,19,644,21,295,11
6,595,904,76,51,703,430,95,408,89
11,495,844,13,417,570,9,429,16,939
430,270,49,72,65,66,338,994,167,76
47,211,87,39,1,570,85,134,967,12

553,63,35,63,98,402,664,85,458,834
3,62,508,7,1,72,88,45,496,43
750,222,96,31,278,184,36,7,210,55
653,51,35,37,393,2,49,884,418,379
75,338,51,21,29,95,790,846,720,71
728,930,95,1,910,5,804,5,284,128

423,6,58,36,37,321,22,26,16,27
218,530,93,55,89,71,828,75,628,67
66,622,440,91,73,790,710,59,83,968
129,632,170,67,613,608,43,71,730,910
36,92,950,138,23,95,460,62,189,73
65,943,62,554,46,318,13,540,90,53

967,654,46,69,26,769,82,89,15,87

46,59,22,840,66,35,684,57,254,230
21,586,51,19,984,156,23,748,760,65
339,892,13,13,327,65,35,246,71,178
83,3,34,624,788,200,980,882,343,550
708,542,53,72,86,51,700,524,577,948

132,900,72,51,91,150,22,110,154,148
99,75,21,544,110,11,52,840,201,2
6,663,22,20,89,10,93,964,924,73
501,398,3,2,279,5,288,80,91,132
620,628,57,79,2,874,36,497,846,22
350,866,57,86,83,178,968,52,399,628

869,26,710,37,81,89,6,82,82,56
96,66,46,13,934,49,394,72,194,408
5,541,88,93,36,398,508,89,66,16
71,466,7,95,464,41,69,130,488,695
82,39,95,53,37,200,87,56,268,71
304,855,22,564,47,26,26,370,569,2

494,2,25,61,674,638,61,59,62,690
630,86,198,24,15,650,75,25,571,338
268,958,95,898,56,585,99,83,21,600
462,940,96,464,228,93,72,734,89,287
174,62,51,73,42,838,82,515,532,91
25,47,12,56,65,734,70,48,209,71

267,290,31,844,12,570,13,69,65,848
72,780,27,96,97,17,69,274,616,36
554,236,47,7,47,134,76,62,824,55
374,471,478,504,496,754,604,923,330,22
97,6,2,16,14,958,53,480,482,93
57,641,72,75,51,96,83,47,403,32

624,7,96,45,97,148,91,3,69,26
22,45,42,2,75,76,96,67,688,2
2,224,83,69,41,660,81,89,93,27
214,458,66,72,384,59,76,538,15,840
65,63,77,33,92,32,35,832,970,49
13,8,77,75,51,95,56,63,578,47

33,63,928,292,2,340,278,911,818,770
464,53,888,55,76,31,389,40,864,36
35,37,69,95,22,648,334,14,198,42
73,594,95,32,814,45,45,515,634,254
42,29,15,83,55,176,35,46,60,296
262,598,67,644,80,999,3,727,79,374

19,780,400,588,37,86,23,583,518,42
56,1,108,83,43,720,570,81,674,25
96,218,6,69,107,534,158,56,5,938
9,938,274,76,298,9,518,571,47,175
63,93,49,94,42,26,79,50,718,926
419,810,23,363,519,339,86,751,7,86

47,75,55,554,3,800,6,13,85,65
99,45,69,73,864,95,199,924,19,948
214,3,718,56,278,1,363,86,1,22
56,114,13,53,56,19,82,88,99,543
674,704,418,670,554,282,5,67,63,466
491,49,67,154,956,911,77,635,2,49

53,12,79,481,218,26,624,954,13,580
130,608,3,37,91,78,743,1,950,45
41,718,36,30,534,418,452,359,759,88
29,499,55,974,93,56,108,257,93,171
13,92,63,714,9,84,890,16,930,967
748,5,7,6,327,894,33,629,448,21

9,19,7,535,75,3,27,928,21,7
864,27,73,61,25,75,876,16,92,22
248,11,86,944,872,996,252,2,800,334
93,107,254,441,930,744,97,177,498,931
694,800,9,36,6,539,35,79,130,860
710,7,630,475,903,552,2,45,97,974

17,36,77,843,328,22,76,368,39,71
35,850,96,93,87,56,972,96,594,864
344,76,17,17,576,629,780,640,56,65
43,196,520,86,92,31,6,593,174,569.
89,718,83,8,790,285,780,62,378,313
519,2,85,845,931,731,42,365,32,33

65,59,2,671,26,364,854,526,570,630
33,654,95,41,42,27,584,17,724,59
42,26,918,6,242,356,75,644,818,168
964,12,97,178,634,21,3,586,47,382
804,89,194,21,610,168,79,96,87,266
482,46,96,969,629,128,924,812,19,2

468,13,9,120,73,7,92,99,93,418
224,22,7,29,57,33,949,65,92,898
200,56,12,31,296,185,272,91,77,37
734,911,27,310,59,33,87,872,73,79
920,85,59,72,888,49,12,79,538,947
462,444,828,935,518,894,13,591,22,920

23,93,87,490,32,63,870,393,52,23
63,634,39,83,12,72,131,69,984,87
86,99,52,110,183,704,232,674,384,47
804,99,83,81,174,99,77,708,7,623
114,1,750,49,284,492,11,61,6,449
429,52,62,482,826,147,338,911,30,984

35,55,21,264,5,35,92,128,65,27
9,52,66,51,7,47,670,83,76,7
79,37,2,46,480,608,990,53,47,19
35,518,71,59,32,87,96,240,52,310
86,73,52,31,83,544,16,15,21,774
224,7,83,680,554,310,96,844,29,61

<= Right Hand Sides (bi) :

10000,10000,10000,10000,10000,10000,10000,10000,10000,7000
10000,10000,10000,10000,10000,10000,10000,10000,10000,7000
10000,10000,10000,10000,10000,10000,10000,10000,10000,7000

Problem 2 [13]

6 variables, 4 constraints

Maximize Obective Function:

500,900,300,600,100,300

Subject to Constraint Matrix:

1,4,5,7,5,6
5,3,2,8,4,9
4,3,4,4,2,5
6,5,3,7,2,5

<= Right Hand Sides (bi):

16,16,16,16

Problem 3 [13]

8 variables, 2 constraints

Maximize Objective Function:

100,400,200,800,300,600,400,500

Subject to Constraint Matrix:

6,2,3,4,9,6,5,1
2,8,2,6,3,5,6,7

<= Right Hand Sides (bi):

24,30

Problem 4 [13]

Variables 60,Constraints 30

Maximize Objective Function:

2,77,6,67,930,3,6,270,33,13
110,21,56,974,47,734,238,75,200,51
47,63,7,6,468,72,95,82,91,83
27,13,6,76,55,72,300,6,65,393
63,61,52,85,29,640,558,53,47,25
3,6,568,6,2,780,69,31,774,22

Subject to Constraint Matrix:

47,774,76,56,59,22,42,1,21,760
818,62,42,36,785,29,662,49,608,116
834,57,42,39,994,690,27,524,23,96
667,490,805,46,19,26,97,71,699,465
53,26,123,20,25,450,22,979,75,96
27,41,21,81,15,76,97,646,898,37

73,67,27,99,35,794,53,378,234,32
792,97,64,19,435,712,837,22,504,332
13,65,86,29,894,266,75,16,86,91
67,445,118,73,97,370,88,85,165,268
758,21,255,81,5,774,39,377,18,370
96,61,57,23,13,164,908,834,960,87

36,42,56,96,438,49,57,16,978,9
644,584,82,550,283,340,596,788,33,350
55,59,348,66,468,983,6,33,42,96
464,175,33,97,15,22,9,554,358,587
71,23,931,931,94,798,73,873,22,39
71,864,59,82,16,444,37,475,65,5

47,114,26,668,82,43,55,55,56,27

716,7,77,26,950,320,350,95,714,789
430,97,590,32,69,264,19,51,97,33
571,388,602,140,15,85,42,66,778,936
61,23,449,973,828,33,53,297,75,3
54,27,918,11,620,13,28,80,79,3

61,720,7,31,22,82,688,19,82,654
809,99,81,97,830,826,775,72,9,719
740,860,72,30,82,112,66,638,150,13
586,590,519,2,320,13,964,754,70,241
72,12,996,868,36,91,79,221,49,960
23,18,748,408,688,97,85,777,294,17

698,53,290,3,62,37,704,810,42,17
983,11,45,56,234,389,712,664,59,15
22,91,57,784,75,719,294,978,75,86
105,227,760,2,190,3,71,32,210,678
41,93,47,581,37,977,62,503,32,85
31,36,30,328,74,31,56,891,62,97

71,37,978,93,9,23,47,71,744,9
619,32,214,31,796,103,593,16,468,700
884,67,36,3,93,71,734,504,81,53
509,114,293,31,75,59,99,11,67,306
96,218,845,303,3,319,86,724,22,838
82,5,330,58,55,66,53,916,89,56

33,27,13,57,6,87,21,12,15,290
206,420,32,880,854,417,770,4,12,952
604,13,96,910,34,460,76,16,140,100
876,622,559,39,640,59,6,244,232,513
644,7,813,624,990,274,808,372,2,694
804,39,5,644,914,484,1,8,43,92

16,36,538,210,844,520,33,73,100,284
650,85,894,2,206,637,324,318,7,556
46,818,92,65,520,721,90,53,174,43
320,812,382,16,878,678,29,92,755,827
27,218,143,12,57,480,154,944,7,730
12,65,67,39,390,32,39,318,47,86

45,51,59,21,53,43,25,7,42,27
310,45,72,53,798,304,354,79,45,44
52,76,45,26,27,968,86,16,62,85
790,208,390,36,62,83,93,16,574,150
99,7,920,860,12,404,31,560,37,32
9,62,7,43,17,77,73,368,66,82

11,51,97,26,83,426,92,39,66,2
23,93,85,660,85,774,77,77,927,868
7,554,760,104,48,202,45,75,51,55
716,752,37,95,267,91,5,956,444,529
96,99,17,99,62,7,394,580,604,89
678,476,97,234,1,608,19,69,67,51

410,89,414,81,130,491,6,238,79,43
5,288,910,204,948,19,644,21,295,11
6,595,904,76,51,703,430,95,408,89
11,495,844,13,417,570,9,429,16,939
430,270,49,72,65,66,338,994,167,76
47,211,87,39,1,570,85,134,967,12

553,63,35,63,98,402,664,85,458,834
3,62,508,7,1,72,88,45,496,43
750,222,96,31,278,184,36,7,210,55
653,51,35,37,393,2,49,884,418,379
75,338,51,21,29,95,790,846,720,71
728,930,95,1,910,5,804,5,284,128

423,6,58,36,37,321,22,26,16,27
218,530,93,55,89,71,828,75,628,67
66,622,440,91,73,790,710,59,83,968
129,632,170,67,613,608,43,71,730,910
36,92,950,138,23,95,460,62,189,73
65,943,62,554,46,318,13,540,90,53

967,654,46,69,26,769,82,89,15,87
46,59,22,840,66,35,684,57,254,230
21,586,51,19,984,156,23,748,760,65
339,892,13,13,327,65,35,246,71,178
83,3,34,624,788,200,980,882,343,550
708,542,53,72,86,51,700,524,577,948

132,900,72,51,91,150,22,110,154,148
99,75,21,544,110,11,52,840,201,2
6,663,22,20,89,10,93,964,924,73
501,398,3,2,279,5,288,80,91,132
620,628,57,79,2,874,36,497,846,22
350,866,57,86,83,178,968,52,399,628

869,26,710,37,81,89,6,82,82,56
96,66,46,13,934,49,394,72,194,408
5,541,88,93,36,398,508,89,66,16
71,466,7,95,464,41,69,130,488,695
82,39,95,53,37,200,87,56,268,71
304,855,22,564,47,26,26,370,569,2

494,2,25,61,674,638,61,59,62,690
630,86,198,24,15,650,75,25,571,338
268,958,95,898,56,585,99,83,21,600
462,940,96,464,228,93,72,734,89,287
174,62,51,73,42,838,82,515,532,91
25,47,12,56,65,734,70,48,209,71

267,290,31,844,12,570,13,69,65,848
72,780,27,96,97,17,69,274,616,36
554,236,47,7,47,134,76,62,824,55
374,471,478,504,496,754,604,923,330,22
97,6,2,16,14,958,53,480,482,93
57,641,72,75,51,96,83,47,403,32

624,7,96,45,97,148,91,3,69,26
22,45,42,2,75,76,96,67,688,2
2,224,83,69,41,660,81,89,93,27
214,458,66,72,384,59,76,538,15,840
65,63,77,33,92,32,35,832,970,49
13,8,77,75,51,95,56,63,578,47

33,63,928,292,2,340,278,911,818,770
464,53,888,55,76,31,389,40,864,36
35,37,69,95,22,648,334,14,198,42
73,594,95,32,814,45,45,515,634,254
42,29,15,83,55,176,35,46,60,296

262,598,67,644,80,999,3,727,79,374

19,780,400,588,37,86,23,583,518,42
56,1,108,83,43,720,570,81,674,25
96,218,6,69,107,534,158,56,5,938
9,938,274,76,298,9,518,571,47,175
63,93,49,94,42,26,79,50,718,926
419,810,23,363,519,339,86,751,7,86

47,75,55,554,3,800,6,13,85,65
99,45,69,73,864,95,199,924,19,948
214,3,718,56,278,1,363,86,1,22
56,114,13,53,56,19,82,88,99,543
674,704,418,670,554,282,5,67,63,466
491,49,67,154,956,911,77,635,2,49

53,12,79,481,218,26,624,954,13,580
130,608,3,37,91,78,743,1,950,45
41,718,36,30,534,418,452,359,759,88
29,499,55,974,93,56,108,257,93,171
13,92,63,714,9,84,890,16,930,967
748,5,7,6,327,894,33,629,448,21

9,19,7,535,75,3,27,928,21,7
864,27,73,61,25,75,876,16,92,22
248,11,86,944,872,996,252,2,800,334
93,107,254,441,930,744,97,177,498,931
694,800,9,36,6,539,35,79,130,860
710,7,630,475,903,552,2,45,97,974

17,36,77,843,328,22,76,368,39,71
35,850,96,93,87,56,972,96,594,864
344,76,17,17,576,629,780,640,56,65
43,196,520,86,92,31,6,593,174,569.
89,718,83,8,790,285,780,62,378,313
519,2,85,845,931,731,42,365,32,33

65,59,2,671,26,364,854,526,570,630
33,654,95,41,42,27,584,17,724,59
42,26,918,6,242,356,75,644,818,168
964,12,97,178,634,21,3,586,47,382

804,89,194,21,610,168,79,96,87,266
482,46,96,969,629,128,924,812,19,2

468,13,9,120,73,7,92,99,93,418
224,22,7,29,57,33,949,65,92,898
200,56,12,31,296,185,272,91,77,37
734,911,27,310,59,33,87,872,73,79
920,85,59,72,888,49,12,79,538,947
462,444,828,935,518,894,13,591,22,920

23,93,87,490,32,63,870,393,52,23
63,634,39,83,12,72,131,69,984,87
86,99,52,110,183,704,232,674,384,47
804,99,83,81,174,99,77,708,7,623
114,1,750,49,284,492,11,61,6,449
429,52,62,482,826,147,338,911,30,984

35,55,21,264,5,35,92,128,65,27
9,52,66,51,7,47,670,83,76,7
79,37,2,46,480,608,990,53,47,19
35,518,71,59,32,87,96,240,52,310
86,73,52,31,83,544,16,15,21,774
224,7,83,680,554,310,96,844,29,61

<= Right Hand Sides (bi) :

6000,6000,6000,6000,6000,6000,6000,6000,6000,4000
6000,6000,6000,6000,6000,6000,6000,6000,6000,4000
6000,6000,6000,6000,6000,6000,6000,6000,6000,4000

Appendix B

General Test Problems

Problem 1 [9]

7 variables, 5 constraints

Maximize Objective Function:

180,190,195,183,178,187,198

Subject to Constraint Matrix:

15,12,19,6,18,23,20
20,5,13,9,26,20,12
9,10,18,0,7,15,23
8,6,8,6,7,12,14
18,11,7,10,19,16,7

<= Right Hand Sides (bi):

90,76,55,43,40

Problem 2 [9]

8 variables, 6 constraints

Maximize Objective Function:

560,300,620,431,68,322,103,420

Subject to Constraint Matrix:

40,10,30,20,3,9,15,21
16,41,16,23,4,12,15,22
38,32,71,26,5,30,14,8
8,30,60,18,6,31,17,31
38,30,42,9,7,21,18,38
33,27,37,18,8,29,10,17

<= Right Hand Sides (bi):

105,117,140,130,140,127

Problem 3 [9]

9 variables, 9 constraints

Maximize Objective Function:

560,1125,300,620,2100,431,68,328,47

Subject to Constraint Matrix:

40,91,25,30,160,20,3,12,5
16,92,41,16,150,23,4,27,6
38,39,32,71,80,26,5,40,8
8,71,30,60,200,18,6,30,4
38,52,30,42,170,9,7,20,6
27,48,25,53,135,30,6,23,5
33,67,40,48,149,25,4,31,6
40,85,35,56,190,24,8,35,7
20,45,18,33,110,15,4,17,3

<= Right Hand Sides (bi):

275,275,170,330,270,245,290,320,180

Appendix C

Petersen Test Problems

Problem 1 [8]

6 variables,10 constraints

Maximixe Objective Function:

100,600,1200,2400,500,2000

Subject to Constraint Matrix:

8,12,13,64,22,41
8,12,13,75,22,41
3,6,4,18,6,4
5,10,8,32,6,12
5,13,8,42,6,20
5,13,8,48,6,20
0,0,0,0,8,0
3,0,4,0,8,0
3,2,4,0,8,4
3,2,4,8,8,4

<= Right Hand Sides(bi):

80,96,20,36,44,48,10,18,22,24

Problem 2 [8]

10 variables,10 constraints

Maximize Objective Function:

600.1,310.5,1800,3850,18.6,198.7,882,4200,402.5,327

Subject to Constraint Matrix:

20,5,100,200,2,4,60,150,80,40
20,7,130,280,2,8,110,210,100,40
60,3,50,100,4,2,20,40,6,12
60,8,70,200,4,6,40,70,16,20
60,13,70,250,4,10,60,90,20,24
60,13,70,280,4,10,70,105,22,28
5,2,20,100,2,5,10,60,0,0
45,14,80,180,6,10,40,100,20,0
55,14,80,200,6,10,50,140,30,40
65,14,80,220,6,10,50,180,30,50

<= Right Hand Sides(bi):

450,540,200,360,440,480,200,360,440,480

Problem 3 [8]

15 variables, 10 constraints

Maximize Objective Function:

100,220,90,400,300,400,205,120,160,580,400,140,100,1300,650

Subject to Constraint Matrix:

8,24,13,80,70,80,45,15,28,90,130,32,20,120,40
8,44,13,100,100,90,75,25,28,120,130,32,40,160,40
3,6,4,20,20,30,8,3,12,14,40,6,3,20,5
5,9,6,40,30,40,16,5,18,24,60,16,11,30,25
5,11,7,50,40,40,19,7,18,29,70,21,17,30,25
5,11,7,55,40,40,21,9,18,29,70,21,17,35,25
0,0,1,10,4,10,0,6,0,6,32,3,0,70,10
3,4,5,20,14,20,6,12,10,18,42,9,12,100,20
3,6,9,30,29,20,12,12,10,30,42,18,18,110,20
3,8,9,35,29,20,16,15,10,30,42,20,18,120,20

<= Right Hand Sides (bi):

550,700,130,240,280,310,110,205,260,275

Problem 4 [8]

20 variables,10 constraints

Maximize Objective Function:

100,220,90,400,300,400,205,120,160,580,400,140,100,1300,650,320,480,80,60,2550

Subject to Constraint Matrix:

8,24,13,80,70,80,45,15,28,90,130,32,20,120,40,30,20,6,3,180
8,44,13,100,100,90,75,25,28,120,130,32,40,160,40,60,55,10,6,240
3,6,4,20,20,30,8,3,12,14,40,6,3,20,5,0,5,3,0,20
5,9,6,40,30,40,16,5,18,24,60,16,11,30,25,10,13,5,1,80
5,11,7,50,40,40,19,7,18,29,70,21,17,30,25,15,25,5,1,100
5,11,7,55,40,40,21,9,18,29,70,21,17,35,25,20,25,5,2,110
0,0,1,10,4,10,0,6,0,6,32,3,0,70,10,0,0,0,0,0
3,4,5,20,14,20,6,12,10,18,42,9,12,100,20,5,6,4,1,20
3,6,9,30,29,20,12,12,10,30,42,18,18,110,20,15,18,7,2,40
3,8,9,35,29,20,16,15,10,30,42,20,18,120,20,20,22,7,3,50

<= Right Hand Sides (bi) :

550,700,130,240,280,310,110,205,260,275

Problem 5 [8]

28 variables,10 constraints

Maximize Objective Function:

100,220,90,400,300,400,205,120,160,580,400,140,100,1300,650,320,480,80,60,2550,3100,
1100,950,450,300,220,200,520

Subject to Constraint Matrix:
8,24,13,80,70,80,45,15,28,90,130,32,20,120,40,30,20,6,3,180,220,50,30,50,12,5,8,18

8,44,13,100,100,90,75,25,28,120,130,32,40,160,40,60,55,10,6,240,290,80,90,70,27,17,8,28

3,6,4,20,20,30,8,3,12,14,40,6,3,20,5,0,5,3,0,20,30,40,10,0,5,0,0,10
5,9,6,40,30,40,16,5,18,24,60,16,11,30,25,10,13,5,1,80,60,50,20,30,10,5,3,20

5,11,7,50,40,40,19,7,18,29,70,21,17,30,25,15,25,5,1,100,70,55,20,50,15,15,6,20
5,11,7,55,40,40,21,9,18,29,70,21,17,35,25,20,25,5,2,110,70,55,20,50,20,15,6,20
0,0,1,10,4,10,0,6,0,6,32,3,0,70,10,0,0,0,0,0,0,30,10,0,10,10,5,0,10

3,4,5,20,14,20,6,12,10,18,42,9,12,100,20,5,6,4,1,20,50,30,5,20,20,10,10,20

3,6,9,30,29,20,12,12,10,30,42,18,18,110,20,15,18,7,2,40,60,50,25,25,25,15,10,28
3,8,9,35,29,20,16,15,10,30,42,20,18,120,20,20,22,7,3,50,60,55,25,30,25,15,10,28

<= Right Hand Sides (Bi) :
930,1210,272,462,532,572,240,400,470,490

Problem 6 [8]

39 variables,5 constraints

Maximize Objective Function:

560,1125,300,620,2100,431,68,328,47,122,322,196,41,25,425,
4260,416,115,82,22,631,132,420,86,42,103,215,81,91,26,49,420,316,72,71,49,108,116,90

Subject to Constraint Matrix:

40,91,10,30,160,20,3,12,3,18,9,25,1,1,10,280,10,8,1,1,49,8,21,6,1,510,8,2,1,0,10,42,6,4,8,0
,10,1

16,92,41,16,150,23,4,18,6,0,12,8,2,1,0,200,20,6,2,1,70,9,22,4,1,5,10,6,4,0,4,12,8,4,3,0,10,0
,6

38,39,32,71,80,26,5,40,8,12,30,15,0,1,23,100,0,20,3,0,40,6,8,0,6,4,22,4,6,1,5,14,8,2,8,0,20,
0,0
8,71,30,60,200,18,6,30,4,8,31,6,3,0,18,60,21,4,0,2,32,15,31,2,2,7,8,2,8,0,2,8,6,7,1,0,0,20,8

38,52,30,42,170,9,7,20,0,3,21,4,1,2,14,310,8,4,6,1,18,15,38,10,4,8,6,0,0,3,0,10,6,1,3,0,3,5,
4

<= Right Hand Sides (bi) :

600,500,500,500,600

Problem 7 [8]

50 variables,5 constraints

Maximize Objective Function:

560,1125,300,620,2100,431,68,328,47,122,322,196,41,25,425,4260,416,115,82,22,631,132
,420,86,42,103,215,81,91,26,49,420,316,72,71,49,108,116,90,738,1811,430,3060,215,58,2
96,620,418,47,81

Subject to Constraint Matrix:

40,91,10,30,160,20,3,12,3,18,9,25,1,1,10,280,10,8,1,1,49,8,21,6,1,5,10,8,2,1,0,10,42,6,4,8,
0,10,140,86,11,120,8,3,32,28,13,2,4

16,92,41,16,150,23,4,18,6,0,12,8,2,1,0,200,20,6,2,1,70,9,22,4,1,5,10,6,4,0,4,12,8,4,3,0,10,0
,6
28,93,9,30,22,0,36,45,13,2,2

38,39,32,71,80,26,5,40,8,12,30,15,0,1,23,100,0,20,3,0,40,6,8,0,6,4,22,4,6,1,5,14,8,2,8,0,20,
0,06,12,6,80,13,6,22,14,0,1,2

8, 71, 30, 60, 200, 18, 6, 30, 4, 8, 31, 6, 3, 0, 18, 60, 21, 4, 0, 2, 32, 15, 31, 2, 2, 7, 8, 2, 8, 0,
2, 8, 6, 7, 1, 0, 0, 20, 8, 14,20,2,40,6,1,14,20,12,0,1

38,52,30,42,170,9,7,20,0,3,21,4,1,2,14,310,8,4,6,1,18,15,38,10,4,8,6,0,0,3,0,10,6,1,3,0,3,5,
4,0,30,12,16,18,3,16,22,30,4,0

<= Right Hand Sides (bi) :

800,650,550,550,650

Appendix D

Knapsack Test Problems

Problem 1 [15]

28 Variables, 2 constraints

Maximize Objective Function:

1898,440,22507,270,14148,3100,4650,30800,615,4975
1160,4225,510,11880,479,440,490,330,110,560,
24355,2885,11748,4550,750,3720,1950,10500

Subject to Constraint Matrix:

45,0,85,150,65,95,30,0,170,0,40,25,20,0,0,25,0,0,25,0,165,0,85,0,0,0,0,100

30,20,125,5,80,25,35,73,12,15,15,40,5,10,10,12,10,9,0,20,60,40,50,36,49,40,19,150

<= Right Hand Sides (bi):
600,600

Problem 2 [15]

28 Variables, 2 constraints

Maximize Objective Function:

1898,440,22507,270,14148,3100,4650,30800,615,4975
1160,4225,510,11880,479,440,490,330,110,560,
24355,2885,11748,4550,750,3720,1950,10500

Subject to Constraint Matrix:

45,0,85,150,65,95,30,0,170,0,40,25,20,0,0,25,0,0,25,0,165,0,85,0,0,0,0,100

30,20,125,5,80,25,35,73,12,15,15,40,5,10,10,12,10,9,0,20,60,40,50,36,49,40,19,150

<= Right Hand Sides (bi):
500,500

Problem 3 [15]

28 Variables, 2 constraints

Maximize Objective Function:

1898,440,22507,270,14148,3100,4650,30800,615,4975
1160,4225,510,11880,479,440,490,330,110,560,
24355,2885,11748,4550,750,3720,1950,10500

Subject to Constraint Matrix:

45,0,85,150,65,95,30,0,170,0,40,25,20,0,0,25,0,0,25,0;165,0,85,0,0,0,0,100

30,20,125,5,80,25,35,73,12,15,15,40,5,10,10,12,10,9,0,20,60,40,50,36,49,40,19,150

<= Right Hand Sides (bi):
300,300

Problem 4 [15]

28 Variables, 2 constraints

Maximize Objective Function:

1898,440,22507,270,14148,3100,4650,30800,615,4975
1160,4225,510,11880,479,440,490,330,110,560,
24355,2885,11748,4550,750,3720,1950,10500

Subject to Constraint Matrix:

45,0,85,150,65,95,30,0,170,0,40,25,20,0,0,25,0,0,25,0,165,0,85,0,0,0,0,100

30,20,125,5,80,25,35,73,12,15,15,40,5,10,10,12,10,9,0,20,60,40,50,36,49,40,19,150

<= Right Hand Sides (bi):
300,600

Problem 5 [15]

28 Variables, 2 constraints

Maximize Objective Function:

1898,440,22507,270,14148,3100,4650,30800,615,4975
1160,4225,510,11880,479,440,490,330,110,560,
24355,2885,11748,4550,750,3720,1950,10500

Subject to Constraint Matrix:

45,0,85,150,65,95,30,0,170,0,40,25,20,0,0,25,0,0,25,0,165,0,85,0,0,0,0,100

30,20,125,5,80,25,35,73,12,15,15,40,5,10,10,12,10,9,0,20,60,40,50,36,49,40,19,150

<= Right Hand Sides (bi):
600,300

Problem 6 [15]

28 Variables, 2 constraints

Maximize Objective Function:

1898,440,22507,270,14148,3100,4650;30800,615,4975
1160,4225,510,11880,479,440,490,330,110,560,
24355,2885,11748,4550,750,3720,1950,10500

Subject to Constraint Matrix:

45,0,85,150,65,95,30,0,170,0,40,25,20,0,0,25,0,0,25,0,165,0,85,0,0,0,0,100

30,20,125,5,80,25,35,73,12,15,15,40,5,10,10,12,10,9,0,20,60,40,50,36,49,40,19,150

<= Right Hand Sides (bi):
562,497

Problem 7 [15]

105 Variables, 2 constraints

Maximize Objective Function:

41850,38261,23800,21697,7074,5587,5560,5500,3450,2391
761,460,367,24785,47910,30250,107200,4235,9835,9262
15000,6399,6155,10874,37100,27040,4117,32240,1600,4500
70610,6570,15290,23840,16500,7010,16020,8000,31026,2568
2365,4350,1972,4975,29400,7471,2700,3840,22400,3575
13500,1125,11950,12753,10568,15600,20652,13150,2900,1790
4970,5770,8180,2450,7140,12470,6010,16000,11100,11093
4685,2590,11500,5820,2842,5000,3300,2800,5420,900
13300,8450,5300,750,1435,2100,7215,2605,2422,5500
8550,2700,540,2550,2450,725,445,700,1720,2675
220,300,405,150,70

Subject to Constraint Matrix:

75,40,365,95,25,17,125,20,22,84
75,50,15,0,0,12,0,10,0,50
0,0,10,0,0,50,60,150,0,0
75,0,102,0,0,40,60,0,165,0
0,0,45,0,0,0,25,0,150,0
0,0,158,0,85,95,0,89,20,0
0,0,0,0,0,80,0,110,0,15
0,60,5,135,0,0,25,0,300,35
100,0,0,25,0,0,225,25,0,0
0,0,0,0,0,5,0,60,0,100
0,0,0,0,0

0,0,0,0,0,0,0,0,0,0
0,0,0,5,10,10,50,2,5,5
10,5,6,11,41,30,5,40,2,6
100,10,25,39,30,13,30,16,60,5
5,10,5,15,91,24,10,15,90,15
60,5,55,60,50,75,100,65,15,10
30,35,50,15,45,80,40,110,80,80
36,20,90,50,25,50,35,30,60,10

150,110,70,10,20,30,104,40,40,94
150,50,10,50,50,16,10,20,50,90
10,15,39,20,20

<= Right Hand Sides (bi):
3000,3000

Problem 8 [15]

105 Variables, 2 constraints

Maximize Objective Function:

41850,38261,23800,21697,7074,5587,5560,5500,3450,2391
761,460,367,24785,47910,30250,107200,4235,9835,9262
15000,6399,6155,10874,37100,27040,4117,32240,1600,4500
70610,6570,15290,23840,16500,7010,16020,8000,31026,2568
2365,4350,1972,4975,29400,7471,2700,3840,22400,3575
13500,1125,11950,12753,10568,15600,20652,13150,2900,1790
4970,5770,8180,2450,7140,12470,6010,16000,11100,11093
4685,2590,11500,5820,2842,5000,3300,2800,5420,900
13300,8450,5300,750,1435,2100,7215,2605,2422,5500
8550,2700,540,2550,2450,725,445,700,1720,2675
220,300,405,150,70

Subject to Constraint Matrix:

75,40,365,95,25,17,125,20,22,84
75,50,15,0,0,12,0,10,0,50
0,0,10,0,0,50,60,150,0,0
75,0,102,0,0,40,60,0,165,0
0,0,45,0,0,0,25,0,150,0
0,0,158,0,85,95,0,89,20,0
0,0,0,0,0,80,0,110,0,15
0,60,5,135,0,0,25,0,300,35
100,0,0,25,0,0,225,25,0,0
0,0,0,0,0,5,0,60,0,100
0,0,0,0,0

0,0,0,0,0,0,0,0,0,0
0,0,0,5,10,10,50,2,5,5
10,5,6,11,41,30,5,40,2,6
100,10,25,39,30,13,30,16,60,5
5,10,5,15,91,24,10,15,90,15
60,5,55,60,50,75,100,65,15,10
30,35,50,15,45,80,40,110,80,80
36,20,90,50,25,50,35,30,60,10

150,110,70,10,20,30,104,40,40,94
150,50,10,50,50,16,10,20,50,90
10,15,39,20,20

<= Right Hand Sides (bi):
500,500

Appendix E

Allocation Test Problems

Problem 1 thru 9 [14]

9 variables, 1 constraint

Maximize Objective Function:

20,18,17,15,15,10,5,3,1,1

Subject to Constraint Matrix:

30,25,20,18,17,11,5,2,1,1 <= bi

where Right Hand side (bi) is:

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

Appendix F

Haldi Test Problems

Problem 1 [4]

15 variables,35 constraints

Minimize Objective Function:

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

Subject to Constraint Matrix:

0,0,1,1,0,1,0,0,0,0,0,0,0,0,0
0,0,0,1,1,0,1,0,0,0,0,0,0,0,0
1,0,0,0,1,0,0,1,0,0,0,0,0,0,0
1,1,0,0,0,0,0,0,1,0,0,0,0,0,0
0,1,1,0,0,0,0,0,0,1,0,0,0,0,0
0,1,0,0,1,1,0,0,0,0,0,0,0,0,0
1,0,1,0,0,0,1,0,0,0,0,0,0,0,0
0,1,0,1,0,0,0,1,0,0,0,0,0,0,0
0,0,1,0,1,0,0,0,1,0,0,0,0,0,0
1,0,0,1,0,0,0,0,0,1,0,0,0,0,0
0,0,0,0,0,0,0,1,1,0,1,0,0,0,0
0,0,0,0,0,0,0,0,1,1,0,1,0,0,0
0,0,0,0,0,1,0,0,0,1,0,0,1,0,0
0,0,0,0,0,1,1,0,0,0,0,0,0,1,0
0,0,0,0,0,0,1,1,0,0,0,0,0,0,1
0,0,0,0,0,0,1,0,0,1,1,0,0,0,0
0,0,0,0,0,1,0,1,0,0,0,1,0,0,0
0,0,0,0,0,0,1,0,1,0,0,0,1,0,0
0,0,0,0,0,0,0,1,0,1,0,0,0,1,0
0,0,0,0,0,1,0,0,1,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,1,1,0
0,1,0,0,0,0,0,0,0,0,0,0,0,1,1
0,0,1,0,0,0,0,0,0,0,1,0,0,0,1
0,0,0,1,0,0,0,0,0,0,0,1,1,0,0
0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0

1,0,0,0,0,0,0,0,0,0,0,1,0,0,1
0,1,0,0,0,0,0,0,0,0,1,0,1,0,0
0,0,1,0,0,0,0,0,0,0,0,1,0,1,0
0,0,0,1,0,0,0,0,0,0,0,0,1,0,1
0,0,0,0,1,0,0,0,0,0,1,0,0,1,0
1,0,0,0,0,1,0,0,0,0,1,0,0,0,0
0,1,0,0,0,0,1,0,0,0,0,1,0,0,0
0,0,1,0,0,0,0,1,0,0,0,0,1,0,0
0,0,0,1,0,0,0,0,1,0,0,0,0,1,0
0,0,0,0,1,0,0,0,0,1,0,0,0,0,1

>= Right Hand Sides (bi) :

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

Problem 2 [4]

15 variables,15 constraints

Minimize Objective Function:

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

Subject to Constraint Matrix:

1,0,0,0,1,1,1,0,0,0,1,1,1,0,1
0,1,0,0,1,0,0,1,1,0,1,1,0,1,1
0,0,1,0,0,1,0,1,0,1,1,0,1,1,0
0,0,0,1,0,0,1,0,1,1,0,1,1,1,1
1,1,0,0,0,1,1,1,1,0,0,0,1,1,0
1,0,1,0,1,0,1,1,0,1,0,1,0,1,0
1,0,0,1,1,1,0,0,1,1,1,0,0,1,0
0,1,1,0,1,1,0,0,1,1,0,1,1,0,0
0,1,0,1,1,0,1,1,0,1,1,0,1,0,0
0,0,1,1,0,1,1,1,1,0,1,1,0,0,0
1,1,1,0,0,0,1,0,1,1,1,0,0,0,1
1,1,0,1,0,1,0,1,0,1,0,1,0,0,1

1,0,1,1,1,0,0,1,1,0,0,0,1,0,1
0,1,1,1,1,1,1,0,0,0,0,0,0,1,1
1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,0

>= Right Hand Sides (bi) :
6,6,6,6,5,5,5,5,5,5,5,4,4,4,4,3

Problem 3 [4]

31 variables,31 constraints

Minimize Objective Function:

1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

Subject to Constraint Matrix:

1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1
1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0
0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1
1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,0
0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0
1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1
0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1
1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1,0
0,1,1,0,0,1,1,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1,1,1,0,0,1,1,0,0
1,1,0,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,1,0,0,1
0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0
1,0,1,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,0,1,0,1
0,1,1,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1,1,1,1,0,0,1,1,0,0,0,0,1,1
1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
0,1,1,0,1,1,1,0,0,1,1,0,0,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0
1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1
0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1
1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,1,0,1,0,0,1,0,1,1,0,1,0,0,1,0,1
0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1
1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0
0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0
1,0,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,1
0,1,1,0,0,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1
1,1,0,0,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,0,1,1,0,0,1,1,0
0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,1,1,1,1
1,0,1,1,0,1,0,1,0,1,0,0,1,0,1,1,0,1,0,0,1,0,1,0,1,0,1,1,0,1,0

0,1,1,1,1,0,0,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,1,1,1,1,0,0
1,1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1

>= Right Hand Sides (bi):

10,10,9,10,9,9,8,10,9,9,8,9,8,8,7,10,9,9,8,9,8,8,7,9,8,8,7,8,7,7,6

Appendix G

THE PROGRAM LISTING

```
THIS ALGORITHM PROVIDES AN OPTIMAL OR GOOD FEASIBLE
 STARTING SOLUTION FOR INTEGER ZERO-ONE PROBLEMS THAT
MAXIMIZE WITH <= CONSTRAINTS OR  MINIMIZE
WITH >= CONSTRAINTS.

 1  INPUT "MAX=1 or MIN=2 PROBLEM?:", Z
LINE 2 READS THE NUMBER OF VARIABLES AND CONSTRAINTS
FROM DATA LINE 400
 2  READ N, M
 3  DIM C(1 TO N)
 4  DIM X(1 TO N)
 5  DIM G(1 TO M)
 6  DIM A(1 TO M, 1 TO N)
 7  DIM A1(1 TO M, 1 TO N)
 8  DIM B(1 TO M)
 9  DIM E(1 TO N)
 10  DIM A2(1 TO N)
 11  DIM PSN(1 TO N)
 12  DIM Y(1 TO N)
LINES 20-30 READ THE MATRIX TO BE INPUT FROM DATA LINE 500
 20  FOR I = 1 TO M
       FOR J = 1 TO N
         READ A(I, J)
         NEXT J
 30  NEXT I
LINE 40 READS THE RIGHT HAND SIDES OF THE CONSTRAINTS FROM
DATA LINE 600
 40  FOR I = 1 TO M
READ B(I)
 50  NEXT I

LINE 55 READS THE OBJECTIVE FUNCTION COEFFICIENTS FROM
DATA LINE 700
 55  FOR J = 1 TO N
```

```
        READ C(J)
 56  NEXT J
LINES 60-70  DIVIDE EACH CONSTRAINT BY ITS RIGHT HAND SIDE
 60  FOR I = 1 TO M
       FOR J = 1 TO N
          A1(I, J) = A(I, J) / B(I)
       NEXT J
 70  NEXT I
LINES 80-90 SUM THE COLUMNS OF THE MATRIX
 80  FOR J = 1 TO N
       FOR I = 1 TO M
          A2(J) = A2(J) + A1(I, J)
        NEXT I
 90  NEXT J
LINES 100-110  CALCULATE "BANG FOR BUCK" EFFICIENCIES
       LET K = 1
       LET P = 2

 100  FOR J = 1 TO N
         E(J) = C(J) * (1 - A2(J) / M)

 110  NEXT J

 115  IF Z = 2 THEN
      GOTO 140
 116  END IF
LINES 120-130 CHOOSE THE NEXT VARIABLE TO ENTER THE
BASIS BASED ON THE BANG FOR BUCK EFFICIENCIES FOR A
MAXIMIZATION PROBLEM
 120  FOR S = 1 TO N - 1
       IF E(K) >= E(P) THEN
          LET K = K
          LET P = P + 1
       ELSE
          LET K = P
          LET P = P + 1
       END IF

 130   NEXT S

     GOTO 160
```

```
LINES 140-150 CHOOSE THE NEXT VARIABLE TO ENTER THE
BASIS BASED ON THE  BANG FOR BUCK EFFICIENCIES FOR A
MINIMIZATION PROBLEM
```

```
140  FOR S = 1 TO N - 1
       IF E(K) <= E(P) THEN
         LET K = K
         LET P = P + 1
       ELSE
         LET K = P
         LET P = P + 1
       END IF
150   NEXT S
```

```
X(K) IS THE NEXT VARIABLE TO ENTER THE BASIS
```

```
160  LET X(K) = 1

     LET COUNTER = COUNTER + 1

170  PSN(COUNTER) = K

     LET T# = 0

190  GOSUB FORMAG

200 IF Z = 2 THEN
      GOTO 240
210 END IF
```

```
LINES 220-230 TEST THE AGGREGATED CONSTRAINT FOR A
MAXIMATION PROBLEM
```

```
220  IF T# <= M THEN
       LET E(K) = -1D+26
       LET K = 1
       LET P = 2
221      IF COUNTER = N THEN
         GOTO 260
222      END IF
       GOTO 120
     ELSE
       GOSUB VAROUT
```

```
      LET X(K) = 0
      LET E(K) = -1D+26
      LET K = 1
      LET P = 2
223       IF COUNTER = N THEN
          GOTO 260
224       END IF
      GOTO 120
230 END IF
```

LINES 240-250 TEST THE AGGREGATED CONSTRAINT FOR A MINIMIZATION PROBLEM AND SET THE OBJECTION FUNCTIONVALUE AS THE UPPER BOUND FOR THE OPTIMAL SOLUTION, "F"

```
240 IF T# <= M THEN
        GOTO 260
    ELSE
      LET E(K) = 10 ^ 8
      LET K = 1
      LET P = 2
      GOTO 140
250 END IF
```

LINES 260-270 COMPUTE THE INITIAL BOUND "G", ON THE OPTIMAL SOLUTION,"F"

```
260 FOR J = 1 TO N
      G = C(J) * X(J) + G
270 NEXT J
```

LINES 275-276 SET THE INITIAL BOUND, "G", AS THE BOUND ON THE OPTIMAL SOLUTION, "F"

```
275 FOR J = 1 TO N
      LET Y(J) = X(J)
    NEXT J
276 LET F = G
    LET G = 0
```

LINES 280-320 ARE THE BACK-TRACKING PROCEDURE TO IMPROVE THE COMPUTED BOUND ON OPTIMAL SOLUTION, “F”

```
280   FOR W = N - 1 TO 0 STEP -1
        LET V = W + 1
```

```
X(K) IS THE NEXT EXIT VARIABLE TO IMPROVE THE LOWER
 BOUND ON "F"
283         LET K = PSN(W)
           IF Y(K) = 0 THEN
             GOTO 320
           END IF
           GOSUB VAROUT
284         LET X(K) = 0
LINES 290-291 CHECK IF THE LAST RECURSION IS COMPLETE
290         IF V = N + 1 AND Z = 1 THEN
              GOTO 310
           ELSE
           IF V = N + 1 AND Z = 2 THEN
              GOTO 320
           END IF
291         END IF
           COUNTER = COUNTER + 1
X(Q) IS THE NEXT ENTRY VARIABLE  TO IMPROVE THE LOWER
BOUND ON "F"

           LET T# = 0
           LET K = PSN(V)
           IF X(K) = 1 THEN
             LET V = V + 1
             GOTO 290
           ELSE
             LET X(K) = 1
           END IF

292         GOSUB FORMAG
 LINES 300-301 TEST IF THE CURRENT VALUES FOR VARIABLES
X(N) SATISFY THE  AGGREGATED CONSTRAINT ,"T"


300         IF T# <= M THEN
             IF Z = 1 THEN
               V = V + 1
               GOTO 290
             ELSE
               GOTO 310
             END IF
```

```
        ELSE
            IF Z = 1 THEN
              GOSUB VAROUT
              LET X(K) = 0
            ELSE
              LET K = PSN(W)
              LET X(K) = 1
              GOSUB VARIN
              GOTO 320
            END IF
            V = V + 1
            GOTO 290
301       END IF

310         FOR J = 1 TO N
            G = C(J) * X(J) + G
        NEXT J
```

LINES 311-315 TEST IF THE CURRENT FEASIBLE SOLUTION,"G" IMPROVES THE LAST COMPUTED LOWER BOUND ON OPTIMAL SOLUTION "F", AND REPLACES "F" IF IT DOES

```
311     IF Z = 1 THEN
        IF G >= F THEN
            FOR J = 1 TO N
              Y(J) = X(J)
              NEXT J
            F = G
         END IF
        END IF
312   IF Z = 2 THEN
        IF G <= F THEN
            FOR J = 1 TO N
              Y(J) = X(J)
              NEXT J
            F = G
         END IF
313   END IF
        LET G = 0

320   NEXT W
```

LINES 330-340 PRINT THE SOLUTION, "F" AND THE VALUES

```
OF THE VARIABLES  Y(N)
330  PRINT " THE VALUES FOR Y ARE :"
     FOR J = 1 TO N
       PRINT "Y"; J; "="; Y(J)
340    NEXT J
     PRINT "The Objective Function Value Is:", F
400 DATA
500 DATA
600 DATA
700 DATA
SUBROUTINE FORMAG FORMS THE CONSTRAINTS INTO AN
AGGREGATED CONSTRAINT,"T#"
1000 FORMAG:
1100  FOR I = 1 TO M
        G(I) = G(I) + A1(I, K)
1200  NEXT I

    LET W# = (10 ^ 10)

    ON ERROR GOTO 1550
1300  FOR I = 1 TO M

         IF Z = 1 THEN
           T# = G(I) ^ W# + T#
         ELSE
          IF G(I) = 0 THEN
            LET T# = M + 1
            GOTO 1500
           END IF
          T# = G(I) ^ -W# + T#
         END IF

1400  NEXT I

1500  RETURN
LINES 1550-1551 IS AN ERROR HANDLING ROUTINE THAT SETS
T#=M+1 WHEN  OVERFLOW OCCURS DUE TO A VIOLATED
CONSTRAINT
1550 LET T# = M + 1
1551    RESUME NEXT
SUBROUTINE VAROUT SUBTRACTS THE EXIT VARIABLE
```

```
CONSTRAINT COEFFICIENT FROM EACH CONSTRAINT
2000 VAROUT:
2100  FOR I = 1 TO M
          G(I) = G(I) - A1(I, K)
2200  NEXT I
2300  RETURN
SUBROUTINE VARIN ADDS THE ENTRY VARIABLE CONSTRAINT
COEFFICIENT TO  EACH CONSTRAINT

3000 VARIN:
3100  FOR I = 1 TO M
          G(I) = G(I) + A1(I, K)
3200  NEXT I
3300  RETURN
```

Appendix H

HOW TO CHOOSE W

W can be chosen for any m constraint zero-one maximization problem based upon a desired tolerance, TOL, using the equation,

$$W > LN(m)/LN(1+TOL) \quad \text{for } m > 1$$

(1)

$$W >= 1 \quad \text{for } m = 1$$

such that no single original constraint /RHS can be larger than 1 + TOL when the aggregated constraint-

$$T\# = (g_1')^W + (g_2')^W + \ldots + (g_m')^W <= m$$

(2)

is tested.

T# is the sum of all original constraints after division by their RHSs.

Derivation of (1)-

IF $(g_i') = 1 + TOL$ then constraint i is violated.

We want W such that $(g_i')^W > m$ when $(g_i') = 1 + TOL$.

$(g_i')^W = (1 + TOL)^W > m$ (by substitution)

$W\ LN(1 + TOL) > LN(m)$ (taking LN of both sides)

$W > LN(m)/LN(1 + TOL)$

Chose W accordingly to allow no LHS of any single original constraint/RHS to be larger

than 1 + TOL.

Note that if (g_i') < 1, then $(g_i')^W$ $<1^W$ =1.

A similar process works for zero-one minimization problems except that

1-TOL is used to calculate W.

EXAMPLE:

Let m=30 ,TOL=.0000000001

Choose W > LN(30)/LN(1.0000000001) = 34011973818.3

ie. W= 34011973819.

$T\# = (g_1')^W + (g_2')^W + \ldots + (g_{30}')^W >= 30.000000002 > m$ is violated if any

one or more g_i' are greater than or equal to 1.000000001.

PROOF (PARTIAL):

CHOOSE W according to the rule W > LN(m)/LN(1 + TOL).

Then if $T\# = (g_1')^W + (g_2')^W + \ldots + (g_m')^W <= m$ no constraints are violated

or if $T\# = (g_1')^W + (g_2')^W + \ldots + (g_m')^W >= m$ some constraints are violated.

Suppose m > 1 and 1 constraint is violated, say constraint 1.

Then $T\# = (g_1')^W + (g_i')^W$

And since $g_1' > 1 + TOL$ then $(g_1')^W > 1$

We know that $T\# > 1 + (g_i')^W$

We also know that $(g_i')^W$ has m-1 terms, each term less than or equal to 1.

Suppose each (g'_i) for i = 2,3,...,m-1 is equal to 1 for all i.

Then $(g_i')^W = m-1$

and $T\# > 1 + (g_i')^W > 1 + (m-1)$

So $T\# > m$.

Hence in this particular case our proof holds.

The above proof appears to hold when any of m constraints are violated. There are however, a number of cases when we could get a false indication when testing the aggregated constraint. For example , if one constraint is just under 1 + TOL and all other constraints are tight, T# would have a value of 30+. This indicates that some constraint is violated. However, because of the desired tolerance, no constraints are violated. We leave these issues for further research.