Generalized Total and Partial Set Covering Problems

by

Edna L. Parrish

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering and Operations Research

APPROVED:

Dr. Prabakar M. Ghare

Dr. Subash C. Sarin

May, 1986
Blacksburg, Virginia
Generalized Total and Partial Set Covering Problems

by

Edna L. Parrish

Dr. Hanif D. Sherali, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

This thesis is concerned with the development of two generalized set covering models. The first model is formulated for the total set covering problem where cost is minimized subject to the constraint that each customer must be served by at least one facility. The second model is constructed for the partial set covering problem in which customer coverage is maximized subject to a budget constraint. The conventional formulations of both the total set covering and partial set covering problems are shown to be special cases of the two generalized models that are developed. Appropriate solution strategies are discussed for each generalized model. A specialized algorithm for a particular case of the partial covering problem is constructed and computational results are presented.
Table of Contents

INTRODUCTION ........................................................................................................... 1
    Problem Description ................................................................................................. 1
    Objectives ................................................................................................................ 2

PROBLEM FORMULATION .......................................................................................... 4
    The Generalized Total Covering Problem .............................................................. 4
    The Generalized Partial Covering Problem ............................................................. 9

SOLUTION TECHNIQUES FOR THE GENERALIZED COVERING PROBLEMS .... 12
    Analysis of the Generalized Partial Covering Model ............................................ 13

LITERATURE REVIEW ............................................................................................... 17
    Set covering algorithms ......................................................................................... 17
    Facility Location Algorithms ................................................................................. 19
    Reliability models ..................................................................................................... 20
    Knapsack problems .................................................................................................. 22
List of Tables

Table 1. Probability Parameters ........................................... 37
Table 2. Strength of Bounding Strategies ......................................... 38
Table 3. Incumbent Solution Strategies ........................................... 39
Table 4. Results of Problem Set I ........................................... 42
Table 5. Results of Problem Set II ........................................... 43
Table 6. Results of Problem Set III ........................................... 44
Table 7. Results of Problem Set II - Depth-First Enumeration ..................... 45
Table 8. Results of Problem Set II - Increased Fathoming Tolerance .............. 45
Table 9. Results of Problem Set I - Hybrid and Depth-First Algorithms ............. 46
I wish to express my deepest and most sincere gratitude to my advisor, Dr. Hanif Sherali. It was his encouragement, patience, and friendship that made the completion of my degree possible. I will always be thankful for the assistance Dr. Sherali has given me, and I feel extremely fortunate to have studied with such a remarkable individual. I would also like to thank the other two members of my committee, Dr. Sarin and Dr. Ghare, for their very valuable assistance and helpful ideas.

I want to express my gratitude to all my friends and family members who have encouraged and supported me while I was working on my graduate degree. I have especially appreciated Judy Paulson, Osman Ulular, and Mark Tokay, who each contributed uniquely to my work during this time through their love, patience, and constant encouragement.

Finally, I would like to dedicate this work to my father, Louis Parrish, and to the memory of my mother, Janet Parrish. They instilled in me a belief in the value of education, and their faith, love, and belief in me have been the most important influences in my life.
INTRODUCTION

The purpose of this chapter is to give a brief introduction to the problem with which this thesis is concerned and to set forth the primary objectives of this investigation. Possible solution strategies for various classes of problems are presented, along with some areas of application for these formulations.

Problem Description

The main subject of this thesis is a particular case of the generalized partial set covering model in which each member of a set of facilities provides probabilistic coverage for each member in a set of customers. The objective of the problem is to maximize the sum of reliabilities of individual customer coverages subject to a budget or resource constraint. The problem is formulated as an integer knapsack problem with a nonlinear reliability type of objective function.

There are two classes of covering problems that are related to the above problem: 1) the total set covering problem that finds a minimum cost solution when all customers must be served by at least one facility, and 2) the partial set covering model where the number of customers covered is optimized subject to a budget constraint.
A number of solution strategies have been proposed for both the total and the partial set covering models in their deterministic forms where all of the covering probabilities are either 0 or 1. Surveys of such algorithms have been published by Garfinkel and Nemhauser (1972) and by Balas and Padberg (1979), and include branch and bound enumeration methods, dynamic programming formulations, and cutting plane techniques. Among the areas of applications listed by these authors for the set covering formulations are:

- crew scheduling
- fleet scheduling
- information retrieval
- political districting
- truck deliveries
- disconnecting paths in a graph
- coloring problems
- cutting stock problems
- line and capacity balancing
- facility location
- capital investment
- switching current design and symbolic logic

In addition, Bausch (1982) lists the following applications:

- PERT-CPM
- frequency allocation
- tracking problems
- nuclear and conventional targeting
- list selection
- tanker routing

**Objectives**

Two generalized covering models will be developed. The first is a Generalized Total Covering model (GTC) in which one minimizes total cost subject to a set of covering reliability type of constraints. The second is a Generalized Partial Covering model (GPC) in which one maximizes a measure of total covering reliability subject to a cost or budgetary type of constraint. The existing formulations of set covering problems will be shown to be special cases of these general models. One special case of the partial set covering model will then be selected for further study since it is
not solvable via the traditional set covering algorithms. Some existing approaches for solving the usual set covering formulations will be reviewed and results of other work that are applicable to the problem addressed herein will be presented. An algorithm will be developed to exploit the special structure of this set covering problem using an enumeration procedure that applies the branch and bound technique. Computational results of the proposed algorithm will be presented, and some possible areas of future research will be suggested.
PROBLEM FORMULATION

The Generalized Total Covering Problem

In this section, we will present a generalized formulation of a class of set covering problems. A set covering problem seeks to satisfy a set of independent demand centers in an optimal manner using a subset of possible supply centers. Within the context of facility location problems, the demand centers are known as customers while the supply centers are called facilities. If there are $r$ various types of facilities, any of which may be situated at $w$ different locations, then $n = r \times w$ possible location-type combinations exist. Each of the $n$ combinations will be referred to as a facility. Associated with each facility is the probability that it provides coverage to customer $i$, and associated with each of the $m$ customers is a minimum coverage requirement. The decision variables specify how many units of each facility should be constructed. The following notation will be used to develop the generalized set covering model:

\[ x_j = \text{number of units of facility } j \text{ that are constructed, where } x_j \geq 0 \text{ and integer} \]

\[ c_j = \text{cost to construct a unit of facility } j \]
\[ p_{ij} = \text{the probability that customer } i \text{ is covered by facility } j \text{ when constructed,} \]
where \( 0 \leq p_{ij} \leq 1 \)

Note that the covering probabilities are assumed to be independent.

\[ p_i = \text{the required coverage reliability for customer } i, \text{ where } 0 < p_i \leq 1 \]

An example of a typical set covering problem for which the generalized formulations presented in this section are necessary is adapted below from a nonlinear programming application described by Kolesar and Walker (1974).

Suppose that crews of firefighters are assigned to certain station houses located throughout a city, and that each crew has an assigned district or area of the city which is its primary responsibility. More than one type of crew may exist (Kolesar and Walker mention engine crews and ladder crews), and a station house may contain several types of crews. Although each crew is assigned primarily to one district, there may be cases where a crew is required to help a crew in another district. However, each district must have some minimum coverage at all times, so a crew is not free to leave its primary area unless some minimum coverage will be maintained by the remaining crews in that district. If it so happens that all crews assigned to a particular area are simultaneously busy, then the closest available crew will be temporarily reassigned to cover that district. Each crew’s coverage of a given district may be assessed in terms of the estimated probability that, when assistance in that district is required, that particular crew is available.

The districts, then, are the customers in the set covering model, while the crews are the facilities. Each of the crews \( j \) has some probability, \( p_{ij} \), of covering customer \( i \), and each customer has some minimum required coverage, \( p_i \). The set covering problem to maximize firefighter coverage over all districts may be formulated several ways, corresponding to the general cases of the total and partial set covering problem discussed below.

The minimum coverage requirements for the \( m \) customers obviously constitute a constraint on the set covering problem; that is, only solutions that provide coverage reliabilities greater than or equal to \( p_i \) for each customer \( i \) will be feasible solutions. Several cases of this type of coverage constraint may be developed as follows:
Denote
\[ I = \{ i \in \{1, \ldots, m\}: p_i = 1 \} \]
as the set of customers for which guaranteed coverage is required and let
\[ S_i = \{ j: p_{ij} = 1 \}, \quad i = 1, \ldots, m \]
denote the set of facilities that are capable of providing guaranteed coverage for customer \( i \).

**Case 1:** Customer \( i \) is an element of set \( I \).

A. If \( S_i = \emptyset \), then the problem is infeasible since no facilities exist that provide guaranteed coverage for customer \( i \).

B. If \( S_i \neq \emptyset \), then there exists some \( x_j \) for \( j \in S_i \) that guarantees coverage for customer \( i \).

Therefore, the set covering constraint is given by
\[ \sum_{j \in S_i} x_j \geq 1. \tag{1} \]

**Case 2:** Customer \( i \) is not an element of set \( I \) and the set \( S_i \) is empty.

Here, absolute coverage is not required for customer \( i \). Hence, the set covering constraint is given by the reliability expression
\[ \prod_{j=1}^{n} (1 - p_{ij})^{x_j} \leq (1 - p_i) \tag{2} \]

which ensures that the combined covering probability of the constructed facilities meets the minimal required covering level for customer \( i \).

Since \( 0 < (1 - p_i) \leq 1 \) and \( 0 < (1 - p_{ij}) < 1 \), taking the logarithms of both sides of equation (2) enables the set covering constraint to be rewritten as
\[ \sum_{j=1}^{n} q_{ij} x_j \geq q_i \tag{2a} \]

where
\[ q_{ij} = - \ln(1 - p_{ij}) \geq 0 \quad \text{and} \quad q_i = - \ln(1 - p_i) > 0. \]
Case 3: Customer \( i \) is not an element of \( I \) and the set \( S_i \) is not empty.

Here, customer \( i \) does not require guaranteed coverage, although there are some facilities that, if constructed, will provide such coverage. If one of the facilities in set \( S_i \) is constructed, then the coverage constraint for customer \( i \) is satisfied. Alternatively, if no facility in set \( S_i \) is chosen, then the coverage constraint given by equation (2a) must be satisfied for customer \( i \). Therefore, either

\[
\sum_{j \in S_i} x_j \geq 1
\]

or

\[
\sum_{j \notin S_i} \left[ \frac{q_j}{q_l} \right] x_j \geq 1
\]

or both constraints should be satisfied for customer \( i \). For integral values of \( x \), this is equivalent to requiring that the following constraint must hold.

\[
\sum_{j \in S_i} x_j + \sum_{j \notin S_i} \left[ \frac{q_j}{q_l} \right] x_j \geq 1
\]  

(3)

The three cases may now be combined to give a general formulation for the total set covering problem. Although constraint (3) subsumes constraints (1) and (2a), for clarity in exposition, let us define

\[
I_1 = \{ i : i \notin I, S_i = \emptyset \}
\]

and

\[
I_2 = \{ i : i \notin I, S_i \neq \emptyset \}
\]

The generalized total covering model (GTC) is as follows:

(GTC):

\[
\text{minimize} \quad z = \sum_{j=1}^{n} c_j x_j \]  

(4a)

subject to

\[
\sum_{j \in S_i} x_j \geq 1, \quad i \in I
\]  

(4b)
The generalized total covering model is a linear integer program which may be solved using one of several techniques. A review of some appropriate algorithms will be included in the following chapter. If a large proportion of the customers are contained in set $I$, it may then be advantageous to devise special procedures to exploit the structure of constraint (4b).

If both sets $I_1$ and $I_2$ are empty, then $p_i = 1$ for all customers $i$. This specialized case of (GTC) is the usual formulation of the total set covering model (TSC), and is as follows:

(TSC):

\[
\text{minimize } z = \sum_{j=1}^{n} c_j x_j \\
\text{subject to } \sum_{j \in S_i} x_j \geq 1, \ i = 1, \ldots, m \\
x_j \geq 0, \text{ integer}
\]  

From the covering constraint (5b), it is apparent that, in any feasible solution, each customer will be served by at least one facility. Since no advantage results in this model from redundant coverage of customers, constraint (5c) is equivalent to the constraint that $x_j$ is binary. Note that in this model, and also in the previous model (GTC), if $c_j = 1$ for all $j$ facilities, then the objective is equivalent to minimizing the number of facilities constructed. Several specialized solution strategies for the total set covering model (5) have been developed and will be reviewed.
The Generalized Partial Covering Problem

The partial covering problem is a converse to the total covering problem in that this problem seeks to maximize the total coverage reliability, subject to a budget constraint which says that the total cost should not exceed some amount $B$. Here, it is recognized that it may not be possible to meet the demands of every customer, i.e. not all the customers may be covered. Hence, the objective is to choose a subset of facilities to construct in order to maximize the total coverage reliability, given the budget limitations.

The general case of the partial covering model will again distinguish between customers that require absolute coverage and those that do not. As with the total covering model, we define

$$S_i = \{ j : p_{ij} = 1 \} , \quad i = 1, \ldots, m$$

and denote all customers that have the potential for absolute coverage as

$$I_3 = \{ i \in \{ 1, \ldots, m \} : S_i \neq \emptyset \} .$$

Also, we define a set of dummy variables

$$y_i = \begin{cases} 1 & \text{if } x_j \geq 1 \text{ for any } j \in S_i \\ 0 & \text{otherwise} \end{cases} \text{ for each } i \in I_3 .$$

The generalized partial covering model becomes:

(GPC):

maximize \[ z = \sum_{i \in I_3} \left[ 1 - \prod_{j=1}^{n} (1 - p_{ij})^{x_j} \right] + \sum_{i \in I_3} \left[ 1 - \prod_{j \in S_i} (1 - p_{ij})^{x_j} (1 - y_i) \right] \] \hfill (6a)

subject to \[ \sum_{j=1}^{n} c_j x_j \leq B \] \hfill (6b)

\[ y_i \leq \sum_{j \in S_i} x_j , \quad i \in I_3 \] \hfill (6c)

\[ x_j \geq 0 \text{ and integer for all } j \] \hfill (6d)

\[ y_i \text{ is binary for all } i \in I_3 \] \hfill (6e)
Note that in model (6), the binary restriction (6e) may be equivalently relaxed to

\[ 0 \leq y_i \leq 1 \text{ for all } i \in I_3. \]  

(6f)

The partial set covering model may be simplified in those cases where all customers \( i \) belong to the set \( I_3 \) and

\[ p_{ij} = 0 \text{ for all } j \notin S_i \]

In this case, the partial set covering model (PSC) is given by:

\begin{align}
\text{(PSC):} & \\
\text{maximize} & \quad z = \sum_{i=1}^{m} y_i \\
\text{subject to} & \quad \sum_{j=1}^{n} c_j x_j \leq B \quad (7b) \\
& \quad y_i \leq \sum_{j \in S_i} x_j \text{ for all } i \quad (7c) \\
& \quad x_j \geq 0, \text{ integer for all } j \quad (7d) \\
& \quad y_i \text{ is binary for all } i \quad (7e)
\end{align}

This problem seeks the maximum number of customers that can be covered in a deterministic coverage situation, given a budgetary constraint, and is converse to the deterministic total set covering problem, (TSC). Consequently, algorithms for the total set covering problem have been adapted to the partial covering model due to this similarity in structure. Furthermore, if \( c_j = 1 \) for all \( j \), then (7b) represents a restriction on the maximum number of facilities (\( B \)) that may be constructed. Also note that constraint (7c) coupled with the objective function guarantees that \( y \) values will be non-fractional if integrality is relaxed, therefore (7e) may be equivalently replaced by

\[ 0 \leq y_i \leq 1. \]  

(7f)
Ignizio (1971) gives a slightly different formulation of the generalized partial covering problem. Rather than maximizing the overall coverage of the customers in the partial covering model, this formulation seeks to maximize the minimum coverage among the customers. This second Generalized Partial Covering model is as follows:

\[(GPCa):\]

\[
\text{maximize } z = \min \left\{ \sum_{i} \sum_{j \in I_i} \left[ 1 - \prod_{j=1}^{n} (1 - p_{ij})^{x_j} \right] + \sum_{i} \sum_{j \not\in S_i} \left[ 1 - \prod_{j \not\in S_i} (1 - p_{ij})^{x_j} (1 - y_i) \right] \right\} \tag{8a}
\]

subject to

\[
\sum_{j=1}^{n} c_j x_j \leq B \tag{8b}
\]

\[
y_i \leq \sum_{j \in S_i} x_j , \quad i \in I_3 \tag{8c}
\]

\[
x_j \geq 0 \text{ and integer for all } j \tag{8d}
\]

\[
y_i \text{ is binary for all } i \in I_3 \tag{8e}
\]

This second formulation of the Generalized Partial Covering problem is expected to be more applicable when the customers are independent or when there is a minimum level of coverage that is desired for each customer. The value of the overall coverage will either remain the same or decrease when this formulation is used rather than the previous model (GPC).

The Literature Review section will discuss several algorithms that have been developed for the Partial Set Covering model.
SOLUTION TECHNIQUES FOR THE
GENERALIZED COVERING PROBLEMS

There are several existing techniques that solve the Generalized Total Covering problem. The branch and bound method of partial enumeration forms the basis for many of these techniques, and dynamic programming has also been applied to smaller problems. Some algorithms developed for the generalized model have been specialized for the Total Set Covering model (5), providing increased efficiency and the ability to solve much larger problems than algorithms for the generalized model can typically handle. A review of these algorithms and their capabilities will be presented in the Literature Review section.

Specialized solution techniques have not been developed, however, for the Generalized Partial Covering model (6). Due to the non-linearity of the objective function, the techniques developed for the Total Covering models are not suitable for the solution of this problem. Therefore, the following analysis of the generalized model is presented to show that it is solvable via known techniques. Following the analysis, some of the appropriate solution techniques will be suggested. Additionally, a special case of the GPC model will be formulated and an algorithm for the solution to this problem will be developed in subsequent chapters.
Analysis of the Generalized Partial Covering Model

First, let us conveniently rewrite (6) by substituting $a_i = (1 - p_i)$ and designating

$$
\tau_i = \begin{cases} 
\prod_{j \in S_i} (a_j)^x_j & \text{if } y_i = 0 \\
0 & \text{if } y_i = 1 
\end{cases} \quad \text{for } i \in I_3.
$$

This reduces the Generalized Partial Covering model (6) to the following:

\begin{align*}
\text{minimize} \quad & z = \sum_{i \in I_3} \prod_{j \in S_i} a_j^{x_j} + \sum_{i \in I_3} \tau_i \\
\text{subject to} \quad & \tau_i \geq \prod_{j \notin S_i} (a_j)^{x_j} - y_i \text{ for all } i \in I_3 \\
& \sum_{j=1}^n c_j x_j \leq B \\
& y_i \leq \sum_{j \in S_i} x_j, \quad i \in I_3 \\
& x_j \geq 0 \text{ and integer for all } j \\
& \tau_i \geq 0 \text{ for all } i \in I_3 \\
& y_i \text{ is binary for all } i \in I_3
\end{align*}

Defining

$$
y_i = \prod_{j \notin S_i} (a_j)^{x_j} \quad \text{and} \quad -z_i = \ln(y_i),
$$

problem (6) may be further simplified to the following:
minimize $z = \sum_{i \in I} \gamma_i + \sum_{i \in I_3} \tau_i$ \hfill (10a)

subject to $\tau_i \geq \gamma_i - y_i$ for all $i \in I_3$ \hfill (10b)

$\sum_{j=1}^{n} c_j x_j \leq \bar{B}$ \hfill (10c)

$y_i \leq \sum_{j \in S_i} x_j, \quad i \in I_3 \hfill (10d)$

$-z_i = \sum_{j \in S_i} x_j \ln(\alpha_j)$ for $i = 1, \ldots, m$ \hfill (10e)

$-z_i = \ln(\eta_i)$ for $i = 1, \ldots, m$ \hfill (10f)

$x_j \geq 0$ and integer for all $j$ \hfill (10g)

$\tau_i \geq 0$ for all $i \in I_3$ \hfill (10h)

$z_i \geq 0$ for $i = 1, \ldots, m$ \hfill (10i)

$y_i$ is binary for all $i \in I_3$ \hfill (10j)

Observe that the integer program (10) has a linear objective function and all the constraints are linear except (10f). The following theorem will enable us to linearize this constraint and rewrite (10) as a linear integer program.

Toward this end, let $\theta_p, p = 1, \ldots, k$, be the values that $z_i$ can possibly take on in (10e), given any integer $x \geq 0$ feasible to (10c), for each $i = 1, \ldots, p$. Then, the required linearization is obtained as follows.

**Theorem 1** Let $\theta_p, p = 1, \ldots, k$, for $i = 1, \ldots, m$ be as defined above. Then problem (10) is equivalent to the linear integer program $P_L$ obtained by replacing (10f) with the constraints

$\gamma_i \geq e^{-\theta_p} [1 + \theta_p - z_i], \quad p = 1, \ldots, k$, for each $i = 1, \ldots, m$. \hfill (11)

**Proof** First of all, note by the convexity of $e^{-z_i}$ that (10f) implies that

$\gamma_i = e^{-z_i} \geq e^{-\theta_p} + (\theta_p - z_i) e^{-\theta_p} = e^{-\theta_p} [1 + \theta_p - z_i]$
or that (11) holds. Hence, it is sufficient to show that at the optimality of the linear integer program

\[ P_L \text{, (10f) automatically holds.} \]

Hence, suppose that \( x^* \) is part of an optimal solution to \( P_L \), and let the constraints (10e) determine \( z_i^* = \theta_{\omega(i)} \) for some \( \rho(i) \in \{1, \ldots, k_i\} \), for each \( i = 1, \ldots, p \). Then from the objective function and from (10b), a corresponding optimal value of \( \gamma \) may be chosen to be as small as possible, subject to (11), namely,

\[
\gamma_i^* = \min_{l = 1, \ldots, k_i} \left( e^{-\theta_{\omega(l)}}[1 + \theta_{\rho(l)} - z_i^*] \right) \text{ for } i = 1, \ldots, p \tag{12}
\]

However, we have by the convexity of the exponential function that

\[
e^{-\theta_{\omega(i)}}[1 + \theta_{\rho(i)} - z_i^*] = e^{-\theta_{\omega(i)}} \geq e^{-\theta_{\rho(i)}} + (\theta_{\rho(i)} - \theta_{\omega(i)})e^{-\theta_{\omega(i)}}
\]

\[
= e^{-\theta_{\rho(i)}}[1 + \theta_{\rho(i)} - z_i^*] \text{ for } i = 1, \ldots, m \tag{13}
\]

From (12) and (13), it therefore follows that \( \gamma_i^* = e^{-\theta_{\omega(i)}} = e^{-z_i^*} \) or that (10f) holds, and the proof is complete. [Q.E.D.]

Of course, the total enumeration of all the constraints in (11) is undesirable, and is not recommended. However, by considering only a subset of (11) and by solving the continuous relaxation of \( P_L \), a lower bound may be computed in a branch and bound algorithm to solve \( P_L \). The partitioning in this branch and bound algorithm may be based on restricting only the \( x \) variables to be integer valued.

In the special case where \( I_3 = \emptyset \), there are no facilities existing that guarantee coverage to any customer, and the Generalized Partial Covering model (6) may be simplified to the following:

\[
(P):
\]

\[
\text{maximize} \quad z = \sum_{l \in I} \prod_{j=1}^{n} c_j x_j \tag{14a}
\]

\[
\text{subject to} \quad \sum_{j=1}^{n} c_j x_j \leq \bar{B} \tag{14b}
\]

\[
x_j \geq 0 \text{ and integer for all } j \tag{14c}
\]
For this problem, traditional integer programming techniques are not appropriate due to the nonlinearity of the objective function and the replacement of the set covering constraint matrix by the knapsack constraint. Knapsack algorithms and algorithms developed for similar types of reliability functions, meanwhile, are not efficient for this problem since they fail to exploit the special set covering structure of the objective function. Therefore, solution techniques for the problem (14) will be the subject of this investigation.
This chapter will present a survey of algorithms that have been developed to solve the specialized cases of the total set covering and partial set covering problems formulated in the previous chapter. These algorithms employ several general purpose techniques such as implicit enumeration, cutting planes, and network-based strategies, and have all been relatively successful in solving the set covering problems. Additionally, reliability models whose objective functions are similar to the objective of the current problem will be presented and the solution techniques that have been successfully applied to these models will be surveyed. Finally, since the current problem has an integer knapsack structure, algorithms for this class of problems will also be discussed.

Set covering algorithms

Implicit enumeration, cutting plane techniques, and network-based solution strategies have all been applied to the set covering problems. The earliest set covering algorithms used branch and bound, or implicit enumeration methods, and this technique continues as one of the most commonly used approaches for solving set covering problems. Balas (1965) developed one of the first implicit enumeration algorithms, employing various logical tests to eliminate several combinations
of variables. Lemke, Salkin, and Spielberg (1971) present a branch and bound methodology for the set covering problem in which they solve linear relaxations of the integer problem in order to derive lower bounds on node subproblems. Branching is achieved by setting the minimal fractional variable from the LP solution to 1 in hopes of getting the largest change in the optimal LP solution for the successive node, and backtracking occurs whenever the LP solution exceeds an incumbent solution value or when the LP solves the integer program.

Marsten (1974) develops a branch and bound technique for the set partitioning problem in which he also solves linear relaxations of the integer problem to find lower bounds on the node subproblems. The bounds are strengthened by employing penalties based on the LP dual multipliers. The main difference, however, between Marsten's algorithm and previous approaches is that he uses a technique known as set partitioning in which variables are grouped together into classes, and certain classes are then selected to cover individual rows. This algorithm compares favorably with both the previous algorithm of Lemke, Salkin, and Spielberg, and with other algorithms designed specifically for the set partitioning problem.

Etcheberry (1977) develops a branch and bound algorithm for the set covering problem that incorporates a set partitioning scheme similar to Marsten's. From each active node, two subnodes are formed by replacing two distinct constraints by one of two valid combinations of these constraints. Associated with the subproblem at each node is a "bounding problem" in which a group of the constraints are dualized via some Lagrange multipliers, and an iterative method of subgradient optimization is used to obtain good values for these Lagrange multipliers.

Etcheberry compares the subgradient algorithm as a bounding technique to linear programming and concludes that a single subgradient iteration takes significantly less computational effort than one simplex pivot iteration, and that the number of subgradient iterations required for any given level of precision does not grow significantly with the size of the problem, while the number of pivots required for the LP solution clearly grows with problem size. Based on the difference in computational effort, Etcheberry concludes that the subgradient optimization technique performs much better than LP as a bounding strategy.
Along with implicit enumeration techniques, several algorithms have been developed in which cutting planes are used. Gomory (1963a, 1963b) was among the first to suggest such a technique where additional constraints are appended to the set covering problem in such a way that some integer solutions are eliminated, reducing the number of feasible solutions that must be considered. Several authors including Lawler (1979), Bellmore and Ratliff (1971), and Balas (1975) have since extended these ideas and proposed branch and bound frameworks within which cutting plane algorithms are found to work well.

In addition to implicit enumeration and cutting plane techniques for solving the set covering problems, there exist a number of network-based solution strategies for these problems. Balas and Padberg (1979) discuss several of these methodologies, and Bausch (1982) computationally tests the network reformulations of Glover, Hultz, Klingman, and Stutz (1978) and of Glover and Mulvey (1980). Bausch also tests several heuristics that significantly improve the computational time of enumeration-based software.

Facility Location Algorithms

The partial covering problem is shown by Ignizio (1971) to be a subclass of a set of problems known as facility location problems. Facility location algorithms are widely used whenever stores, plants, distribution centers, and so forth must be optimally located in order to serve a set of demand points, or customers. Among these algorithms are heuristic techniques that usually make an assignment of facilities to locations and then perturb the solution to see if improvements exist. Several heuristic procedures to solve the partial covering problem are presented by Smallwood (1965), Kuehn and Hamberger (1963), and Ignizio (1971).
Reliability models

Techniques for optimization of reliability-based criteria have been developed for several problems with nonlinear objective functions similar to the problem considered here. Typically, the reliability models may be classified according to their objective functions into parallel redundancy, stand-by redundancy, and combination series-parallel redundancy models. Many of the algorithms used to solve these reliability allocation problems utilize standard integer programming techniques. One of the earliest algorithms, given by Lawler and Bell (1966), rewrites monotone objective functions and constraints as differences in monotone non-decreasing functions. After ordering the variables, the authors use a series of logical tests to eliminate as many solutions as possible. The algorithm then proceeds to enumerate the remaining feasible solutions in a manner similar to branch and bound. Mizukami (1968) presents an algorithm based on convex and integer programming to find optimal redundancy for maximum system reliability. He linearizes the objective function (the constraints are already linear) and then converts the problem to a convex program that may be approximated by a linear program. The most effective of Mizukami's solution methods uses linear programming to get an approximate solution, and then finds an optimal solution using a cutting plane procedure. Tillman and Lüttschauger (1967) solve an n-stage redundancy problem by formulating the integer programs that result from linearizing the objective function for both maximum reliability and minimum cost problems. They additionally solve the optimal parallel design problem by similar linearization techniques. In all three problems, Tillman and Lüttschauger note that the objective functions and the constraints are separable and need not satisfy any convexity or concavity conditions. The implicit enumeration technique of Lawler and Bell was extended by Misra (1971) to solve the redundancy allocation problem for any type of separable constraints; Misra formulates the problem to minimize cost given that the system reliability must be at least some minimum prescribed value, and he also formulates the problem of maximizing system reliability subject to arbitrary constraints. A second algorithm developed by Misra (1972) uses Newton's method with Lagrange multipliers to solve the maximum system reliability problem.
subject to linear constraints. Although this method does not assure an optimal solution, it is good for rapid convergence. A complementary algorithm based on the "maximum principle" approach does guarantee an optimal solution.


Misra and Sharma (1973) employ a branch and bound scheme to solve for maximum reliability of series-parallel systems subject to linear constraints. The authors linearize the objective function and develop methods for choosing branching variables and calculating lower bounds. Luus (1972) deviates somewhat from the enumeration techniques by relaxing integrality for the reliability maximization problem and then rounding all solution values down. He calculates the resulting slack and then systematically exchanges variables to get a good solution. This method, although quick, does not guarantee an optimal solution. Yet another enumeration approach is suggested by Nakagawa, Nakashima, and Hattori (1978) for both the maximum system reliability problem subject to nonlinear resource constraints and the minimum cost problem subject to nonlinear reliability constraints. This algorithm does not require separability in the objective function or constraint functions, and relies on relaxation techniques for enumeration tree bounds. A separation technique similar to that used by Cabot (1970) and McLeavy and McLeavy (1976) is extended here to the nonlinear case in order to reduce the number of nodes that must be considered during the enumeration procedure.
Knapsack problems

The formulation of the current problem indicates that it belongs to the general class of bounded integer knapsack problems. Due to the wide applicability of this class of problem, there are a number of algorithms designed specifically for its solution. Again, some of the techniques that have been applied to the knapsack problems will be useful in developing a solution methodology for the current problem; therefore, a brief review of knapsack algorithms is presented below.

Balas’ implicit enumeration approach (1965) was one of the earliest algorithms applied to integer knapsack problems. Cabot (1970) develops another implicit enumeration approach in which he uses a feasible starting solution and then solves for a solution to a set of inequalities generated by requiring a solution better than the incumbent. A Fourier-Motzkin method of elimination is used to reduce the number of inequalities that must be solved. Cabot’s method is shown to compare favorably with dynamic programming algorithms and other methods of implicit enumeration. Fayard and Plateau (1975) develop a “modified shrinking bound” approach based on the earlier work of Saunders and Schinzinger, and compare their algorithm with the Greenberg and Hegerich (1970) knapsack algorithm and the general integer programming algorithms of Faure (1964) and Geoffrion (1969). Fayard and Plateau concluded that the modified shrinking bound and the Greenberg-Hegerich algorithms were the best overall performers.

For separable, nonlinear, multidimensional knapsack problems, Morin and Marsten (1976) develop a dynamic programming approach. This algorithm accommodates up to 10 dimensional state spaces, avoiding the traditional “curse of dimensionality” that usually impedes dynamic programming approaches to problems of this size. Nauss (1976) presents an improvement on the Horowitz-Sahni (1974) algorithm that includes an ordering of the variables to get lower bounds and uses LP relaxations of the integer problem to find dual multipliers for obtaining surrogate constraints. Martello and Toth (1978) develop a branch and bound algorithm that compares favorably with both the Horowitz-Sahni and the Nauss methods, based on a priority ordering of the decision variables.
Armstrong, Cook, and Palacios-Gomez (1979) formulate a branch and bound methodology for discontinuous nonlinear knapsack problems in which variables must either have a binary value or must lie within a certain interval. This formulation solves a class of problems that has applicability for capital budgeting where some variation in expenses may be expected.

For linear, separable nonlinear, and multiple-choice knapsack problems, Lawler (1979) develops an improved variable scaling procedure, a lower bound that is tighter than traditional LP relaxations, and a median-finding routine that eliminates variable sorting. These techniques are all useful within a basic knapsack optimization routine.

Brown (1979a) develops a variation on the traditional knapsack objective by seeking to maximize the smallest tradeoff function to achieve equitable resource sharing; i.e., he tries to maximize the smallest proportion of resource to weight. Brown calls this problem the “resource sharing problem” and uses a max flow algorithm for its solution, developing tradeoff functions for linear return functions. In a separate paper, Brown (1979b) extends the knapsack sharing methodology to include piecewise linear and nonlinear return functions.

Balas and Zemel (1980) use linear relaxations to solve knapsack problems by focusing on a “core” or subset of variables whose cost-to-weight ratios lie within a certain range. The authors develop a heuristic approximation to the solution of the core problem, and also develop a binary search method to eliminate the time-consuming sorting of the variables. Zemel (1981) later presents a method of measuring the quality of approximate solutions that enforces an equitable measure between solutions of equivalent problems.
This section describes the development of a specialized solution technique for the nonlinear, integer knapsack problem given by (13). Each aspect of the solution methodology is presented along with computational results. The FORTRAN source code for the final algorithm is given in Appendix A.

The branch and bound technique that will be employed in this algorithm is based on Dakin's (1965) method of tree enumeration. Here, starting at the root node or node zero (the beginning) of an enumeration tree, some relaxation of the problem is solved in order to derive a lower bound. Additionally, a quick heuristic technique may be employed in order to derive a good incumbent solution which provides an upper bound. If the relaxed problem is unable to provide an optimal solution to the problem, some integer-restricted variable \( x_s \) is judiciously chosen and the feasible region is partitioned into solutions that restrict \( x_s \leq \delta_s \) and \( x_s \geq \delta_s + 1 \), where \( \delta_s \) is some suitable integer for which \( 0 \leq \delta_s \leq \delta_s + 1 \leq u_s \), the upper bound on \( x_s \). Each of these subproblems represents a new node in the enumeration tree. The lower bound for each of the new nodes may again be determined via some relaxation of the associated subproblem. A node is eliminated from further consideration, or fathomed, whenever its lower bound is worse than the best known solution.
value, which includes the case where the subproblem is infeasible, or when the solution for the relaxation of the problem turns out to provide an optimal solution for the node subproblem. Otherwise, the node is further partitioned into two new subproblems and the process is repeated. This iterative method of partitioning, or branching, and solving subproblems to compute a lower bound continues until the optimum integer solution is found.

The main components, then, of any branch and bound strategy are the methods for computing the lower bounds for each subproblem and the rules governing the choice of branching variables. This chapter describes the lower bounding strategies and the branching rules that comprise the proposed algorithm. A formulation for the subproblem associated with each node in the enumeration tree will first be given based on the original problem formulation (13) given previously.

### Node subproblem formulation

At each node in the branch and bound tree, the associated subproblem is the original problem (13) along with the additional restrictions placed on each branch in the path from that node to the root node. Thus the subproblem at each node has the following form:

\[
\text{(NP):} \quad \begin{align*}
\text{minimize} \quad z &= \sum_{i=1}^{n} \prod_{j=1}^{n} (a_{ij})x_j \\
\text{subject to} \quad \sum_{j=1}^{n} c_jx_j &\leq \bar{B} \\
\bar{t}_j &\leq x_j \leq \bar{u}_j \text{ and integer, } j = 1, \ldots, n
\end{align*}
\]

where \( \bar{t}_j \) and \( \bar{u}_j \) represent the upper and lower bounds imposed on variable \( x_j \) through the branching process leading to this particular node.

The upper bounds can be tightened for each variable by ensuring that, in the above problem, the upper bounds satisfy
\[
\tilde{u}_j \leq \left[ \frac{\bar{B} - \sum_{k \neq j}^n c_k \bar{\ell}_k}{c_j} \right] - , \quad j = 1, \ldots, n
\]

where \([ \cdot ]^-\) denotes the rounded down integer value.

At any node, if \(\bar{\ell}_j = \tilde{u}_j\), then \(x_j\) must equal this common value and may be eliminated from the problem. Furthermore, if \(\bar{\ell}_j > \tilde{u}_j\) for any \(x_j\), the problem is infeasible and the node is fathomed. Otherwise, let

\[
J = \{ j : \tilde{u}_j > \bar{\ell}_j \}
\]

\[
u_j = \tilde{u}_j - \bar{\ell}_j \quad \text{for} \quad j \in J
\]

\[
B = \bar{B} - \sum_{j=1}^n c_j \bar{\ell}_j
\]

and \(\beta_i = \prod_{j=1}^n a_{ij} \bar{\ell}_j / \prod_{j \in J} \alpha_{ij}\) for \(i = 1, \ldots, m\).

Substituting \(y_j = x_j - \bar{\ell}_j\) for each \(j = 1, \ldots, n\) leads to the following equivalent formulation of the node subproblem (15).

\[
\text{(NP1)}:
\]

\[
v(NP1) = \min \sum_{i=1}^m \beta_i \prod_{j \in J} (a_{ij}) y_j
\]

subject to \(\sum_{j \in J} c_j y_j \leq B\) \hspace{1cm} (16b)

\[
0 \leq y_j \leq \tilde{u}_j \text{ and integer,} \quad j = 1, \ldots, n
\]

Of course, if \(B < 0\), then (NP1) is infeasible. Otherwise, we will calculate lower bounds on \(v(NP1)\) via one of the following lower bounding strategies. In those cases where an integer solution that solves the node subproblem is found, the node may be fathomed. Additionally, the node may be fathomed whenever the value of the lower bound is greater than the value of some known solution.
Lower bounding strategies

Four techniques for calculating a lower bound for the node subproblem have been developed. In addition to providing the best possible bound values at each node of the enumeration tree, each lower bounding technique is measured by its ability to provide a good incumbent solution both at node zero and at each subsequent node. If good incumbent solutions are found early on in the enumeration process, more nodes will be fathomed, leading to less overall enumeration.

One of the following bounding techniques provides an integer solution at every node, so the incumbent solution may be updated whenever this bound produces an improved solution. Associated with each of the other bounding strategies is a heuristic rule for finding an integer solution from the continuous bound solution. The integer solution produced by the heuristic rule is then used to update the incumbent solution whenever possible.

Lower Bound 1. Arithmetic vs. geometric mean

The first method of calculating lower bounds for the node subproblem is based on the fact that, for any set of nonnegative numbers $t_1, \ldots, t_m$, the arithmetic mean is greater than or equal to the geometric mean, i.e.

$$\sum_{i=1}^{m} t_i \geq m \left( \prod_{i=1}^{m} t_i \right)^{1/m} \quad (17)$$

Substitution of

$$t_i = \beta_i \prod_{j \in J} (a_{ij})^{y_j} \text{ for } i = 1, \ldots, m$$

in equation (17) gives

$$\sum_{i=1}^{m} \beta_i \prod_{j \in J} (a_{ij})^{y_j} \geq m \left( \prod_{i=1}^{m} \beta_i \prod_{j \in J} (a_{ij})^{y_j} \right)^{1/m}.$$

Therefore, a lower bound may be found for the node subproblem by solving the problem

$$\text{LB1} = \minimizem \left[ \prod_{i=1}^{m} \beta_i \prod_{j \in J} (a_{ij})^{y_j} \right]^{1/m} \quad (18a)$$
subject to \( \sum_{j \in J} c_j y_j \leq B \) \hspace{1cm} (18b)

0 \leq y_j \leq u_j \text{ and integer for all } j \in J \hspace{1cm} (18c)

Note that the term

\[ m \left[ \prod_{l=1}^{m} \beta_l \right]^{1/m} \]

may be treated as a constant for optimization purposes. Also, when all \( t_i \) terms are equal, equation (17) holds as an equality. Therefore, a stronger lower bound value will result when the covering probabilities for each facility are nearly equal for all the \( i \) customers.

The solution of problem (18) is simplified by rewriting the objective function in a linear form.

Denote

\[ \lambda_j = -\frac{1}{m} \log \left[ \prod_{l=1}^{m} \alpha_{jl} \right] \text{ for } j = 1, \ldots, n \] \hspace{1cm} (19)

Hence, the following integer knapsack problem provides an equivalent solution to problem (18).

\[ \text{LB1} = \max \sum_{j \in J} \lambda_j y_j \hspace{1cm} (20a) \]

subject to \( \sum_{j \in J} c_j y_j \leq B \) \hspace{1cm} (20b)

0 \leq y_j \leq u_j \text{ and integer for all } j \in J \hspace{1cm} (20c)

Thus, a lower bound for the node is computed by solving (20) and substituting the antilog of the negative of its objective value into the objective function of (18). Since the \( \lambda_j \) values depend only on the quantities \( \alpha_{jl} \), they may be computed once at the start of the algorithm and used throughout the enumeration procedure.

The problem (20) is a linear integer knapsack problem solvable via a number of techniques that have been previously discussed. The method used in this investigation to solve the integer knapsack problem is a branch and bound technique in which lower bounds at each subnode are
obtained by solving continuous knapsack relaxations. The integer solution that results from solving (20) is used to update the incumbent solution if it offers an improved objective value.

An alternative to solving the integer knapsack is to relax the integrality restriction on (18), since the linear programming solution will give a lower bound on the integer solution. With no integrality restriction, (20) becomes a simple linear bounded knapsack problem which can be solved in $O(|J|)$ time. However, this strategy gives a weaker bound. Any bound between the linear continuous and integer knapsack bounds may be found by performing a partial enumeration. If a partial enumeration is performed, then the best incumbent solution from the knapsack routine may be used to possibly update the incumbent solution of the main program. If the continuous relaxation of (20) is solved, the following heuristic algorithm based on Taha (1975) may be used to possibly update the incumbent solution.

Heuristic Rule 1

Step 1: Round down each fractional $y_j$ and let $\bar{y}$ be the resulting integer solution.

Define $R = \{ j: \bar{y}_j < u_j \}$.

Step 2: Find $k \in R$ with the largest ratio of $\sigma_a$, where $\sigma_a = \frac{\lambda_a}{c_a}$ and $\lambda_a$ is the objective function coefficient of $x_a$ in (20).

Step 3:

If $B - \sum_{j \in J} c_j \bar{y}_j \geq c_k$, then it is possible to increase $y_k$. In this case, let

$$y_k = \min_{k \in R} \left[ \frac{B - \sum_{j \in J} c_j \bar{y}_j}{c_k} \right] , u_k$$

where $[\cdot]$ denotes the rounded down value.

Step 4: Eliminate $k$ from the set $R$.

If $B - \sum_{j \in J} c_j \bar{y}_j < \min_{j \in J} [c_j]$ or if $R = \emptyset$, then stop.
Otherwise, go to step 2.

This heuristic rule quickly produces an integer solution that may then be tested for possible improvement against the incumbent solution. Since any feasible continuous solution may be used as the starting solution for Heuristic Rule 1, this procedure may be used in conjunction with any lower bounding strategy that produces a non-integer solution for the relaxation of the node subproblem.

**Lower Bound 2. Minimum of sum vs. sum of minimum**

A second lower bounding strategy is derived from the fact that the minimum of a sum of terms exceeds the sum of the minimum values of the individual terms, over a specified feasible region. Thus,

\[
\text{minimum } \sum_{i=1}^{m} \beta_i \prod_{j \in J} (\alpha y_j) \geq \sum_{i=1}^{m} \text{minimum } \beta_i \prod_{j \in J} (\alpha y_j)
\]  

This result separates the node problem (16) into \( m \) component problems that are more easily solved. For each of the \( i \) customers, the partial node subproblem is formulated as:

\[
\text{LB2i} = \text{minimum } \beta_i \prod_{j \in J} (\alpha y_j) \]  

subject to \[
\sum_{j \in J} c_j y_j \leq B \]  

\[
0 \leq y_j \leq u_j \text{ and integer for all } j \in J
\]

The lower bound LB2 for the node will equal the sum of the LB2i values found for each of the \( i \) partial problems (22). As in the first lower bounding strategy, the problem (22) may be equivalently solved as a linear knapsack problem by minimizing the log of the objective function. Denote

\[
\lambda y = -\log(\alpha y), \quad i = 1, \ldots, m, \quad j = 1, \ldots, n
\]

and note that
Each problem (22) may now be rewritten as an integer knapsack problem as follows:

\[
\text{maximize} \quad \sum_{j \in J} \lambda_j y_j \quad (23a)
\]

\[
\text{subject to} \quad \sum_{j \in J} c_j y_j \leq B \quad (23b)
\]

\[
0 \leq y_j \leq u_j \text{ and integer for all } j \in J \quad (23c)
\]

The objective value \(LB_{2i}\) of each problem (22) is computed as \(\beta_i\) times the antilog of the negative of the optimal objective value of (23) and the lower bound, \(LB_2\), for the node subproblem is given by the sum of the objective values \(LB_{2i}\).

One feature of this second lower bounding strategy is that it has the potential to solve the node subproblem itself. Observe that if \(y^*\) is an optimal solution to (22) and if it turns out that \(y'' = y^*\) for each \(i = 1, \ldots, m\), then clearly this common solution \(y^*\) also solves (16).

Again, an alternative to solving the set of problems (23) as integer knapsack formulations is to relax the integrality restrictions in (22) and derive quicker, though weaker, lower bounds. A second alternative is to perform a partial enumeration. Either of these may be especially attractive if the magnitude of \(m\) is large.

If \(LB_2\) is computed by solving the continuous relaxation of (22), an integer solution for use in updating the incumbent may be found through use of Heuristic Rule 1 described previously, or by solving the following integer knapsack problem, based on the mean solution

\[
\bar{y}^* = \frac{1}{m} \sum_{i=1}^{m} y_i^*
\]

to the set of problems (LB_{2i}).

Heuristic Rule 2

\[
\text{maximize} \quad z = \sum_{j \in J} \lambda_j y_j \quad (24a)
\]
subject to \[ \sum_{j \in J} c_j y_j \leq B \] \[ \lceil \bar{y}_j \rceil^{-} \leq y_j \leq \bar{u}_j \text{ and integer, } j \in J \]

The solution to Heuristic Rule 2 is an integer feasible solution to (16) and may be used to possibly update the incumbent solution. Again, this heuristic rule may be used with any of the lower bounding strategies that calculate a lower bound by relaxing integrality restrictions.

**Lower Bound 3. Bounds from the continuous relaxation**

A third lower bounding strategy makes use of the fact that the objective value of (16) is convex and the constraints are linear. In this case, the Kuhn-Tucker conditions are both necessary and sufficient for optimality of the continuous relaxation to the node subproblem (16). The continuous relaxation of (16) is given below.

\[(NP1) : \]

\[ \text{LB3 = minimum } \sum_{i=1}^{m} \beta_i \prod_{j \in J} (a_{ij}) y_j \]

subject to \[ \sum_{j \in J} c_j y_j \leq B \]

\[ 0 \leq y_j \leq u_j, \quad j = 1, \ldots, n \]

One method for finding a solution to (25) is to use the feasible direction finding approach of Zoutendijk (1979) as modified by Topkis and Veinott (1979) to ensure convergence. First, note that since \( c_j > 0 \) for all \( j \) and since the objective function of (16) is monotone decreasing in each \( y_j \), when the integrality restrictions are relaxed in (16), the budget constraint must hold as an equality at optimality, provided \( B < \sum_{j \in J} c_j u_j \). Otherwise, \( y_j = u_j \) for all \( j \) is optimal in (16). Hence, given a feasible solution \( \bar{y} \) to (25), the direction finding problem seeks a direction \( d \) which solves the following problem.
(DF):

\[
\begin{align*}
\text{minimize} & \quad \nabla f(\bar{y})'d \\
\text{subject to} & \quad c \cdot d = 0 \\
& \quad -\bar{y}_j \leq d_j \leq u_j - \bar{y}_j, \quad j \in J
\end{align*}
\]

where

\[
f(y) = \sum_{l=1}^{m} \beta_l \prod_{j \in J} (a_{lj})^{y_j}
\]

and \( \nabla f(\bar{y}) \) is the gradient of \( f(\cdot) \) at \( y = \bar{y} \).

Substituting

\[x_j = \bar{y}_j + d_j, \quad j = 1, \ldots, n\]

and denoting

\[
\nabla J = -\left. \frac{\partial f}{\partial y_j} \right|_{y = \bar{y}} > 0,
\]

the model (DF) may be rewritten as the following equivalent direction finding problem:

(EDF):

\[
\begin{align*}
\nu(\bar{y}) = \sum_{j \in J} \nabla J \bar{y}_j - \text{maximum} & \quad \sum_{j \in J} \nabla J x_j \\
\text{subject to} & \quad \sum_{j \in J} c_j x_j = B \\
& \quad 0 \leq x_j \leq u_j, j \in J
\end{align*}
\]

Now the direction finding problem is easily solvable, since (EDF) is a continuous linear bounded variable knapsack problem.

If the optimal solution to the direction finding problem is 0, then \( \bar{y} \) is an optimal solution to (16). Otherwise, since \( \bar{y} \) is feasible in (27), \( \nu(\bar{y}) < 0 \). This means that an improving feasible direction \( d \) exists, and is given by \( d = (\bar{x} - \bar{y}) \) where \( \bar{x} \) solves EDF. Furthermore, a maximum step length along \( d \) which maintains feasibility is unity. A line search carried out along the segment
(0, 1] will locate the minimum of \( f(\tilde{y} + \lambda d) \) over this interval. The solution \( \tilde{y} \) can then be updated accordingly. The algorithm proceeds in this iterative manner until the optimal solution to the continuous relaxation of the node subproblem is found. If the optimal solution is integer, then the subproblem (16) is solved and the node may be fathomed. Otherwise, the continuous optimal solution value gives a lower bound for the node subproblem.

Since an exact solution to \( \overline{NP} \) including exact line searches may be time consuming, an inexact line search technique was employed, along with an \( \varepsilon \) termination criterion. Note that since \( f(\cdot) \) is convex, if \( y^* \) solves \( \overline{NP} \) and if we terminate the algorithm with some feasible solution \( \tilde{y} \), then since \( f(y^*) \geq f(\tilde{y}) + \nabla f(\tilde{y})(y^* - \tilde{y}) \), and since \( v(\tilde{y}) \leq \nabla f(\tilde{y})(y^* - \tilde{y}) \) a valid lower bound is given by adding \( v(\tilde{y}) \) to \( f(\tilde{y}) \). Hence, the algorithm may be terminated when \( -v(\tilde{y}) \) is small enough, and one may use \( v(\tilde{y}) + f(\tilde{y}) \) as LB3.

The inexact line search technique employed here is the quadratic interpolation method, where objective function evaluations are curtailed as soon as three points \((\lambda_1, \theta_1), (\lambda_2, \theta_2), (\lambda_3, \theta_3)\), where \( \lambda_1 < \lambda_2 < \lambda_3 \) are found while systematically increasing \( \lambda \), such that \( \theta_1 \geq \theta_2 \) and \( \theta_2 \leq \theta_3 \). A quadratic curve is fitted to these three points, and the minimum value \( \lambda^* \) of the curve is calculated using the closed form derivative. This \( \lambda^* \) gives the chosen step size.

Since the solution to (27) provides a non-integer feasible solution to the relaxation of (16), either of the two heuristic rules presented previously may be used to find an integer solution that may update the incumbent.

Lower Bound 4. Linear discretization

The fourth lower bounding strategy is derived by rewriting (16) as an equivalent program with a linear objective function using Theorem 1. Defining

\[
x_i = \prod_{j \in J} \alpha_{ij}^{y_j}.
\]

and letting

\[
z_i = \ln(x_i) = \sum_{j \in J} y_j \ln(\alpha_{ij}),
\]

the original node subproblem (16) may be equivalently rewritten as follows by Theorem 1:

SOLUTION STRATEGY FOR A SPECIAL CASE
minimize \( z = \sum_{l=1}^{m} \beta_l x_l \) \hspace{1cm} (28a)

subject to \( \sum_{j \in J} c_j y_j \leq \overline{B} \) \hspace{1cm} (28b)

\( -z_i = \sum_{j \in J} x_j \ln(\alpha_{ij}) \), \( i = 1, \ldots, m \) \hspace{1cm} (28c)

\( \alpha_i \geq e^{-\theta_p \left[ 1 + \theta_{ip} - z_i \right]}, p = 1, \ldots, k_i, i = 1, \ldots, m \) \hspace{1cm} (28d)

\( 0 \leq y_j \leq u_j, \text{ and integer } j = 1, \ldots, n \) \hspace{1cm} (28e)

\( z_i \geq 0 \text{ for all } i \) \hspace{1cm} (28f)

Here again, \( \theta_p, p = 1, \ldots, k_i \) are the possible values that \( z_i \) can take on in (28c), when \( y \) satisfies (28b) and (28c), for each \( i = 1, \ldots, m \). Since there may be too many constraints in (28d), one may relax this problem by selecting

\( \{ 0 < \theta_{i1} < \theta_{i2} < \ldots < \theta_{iq_i} < 1 \} \), \text{ for each } i = 1, \ldots, m \)

and by writing (28d) only for \( p = 1, \ldots, q_i \) for each \( i = 1, \ldots, m \). The continuous relaxation of this problem can now be solved as a linear program in order to obtain a lower bound, LB4, say.

The choice of points for \( \theta_p \) may be based on the values expected for \( x_i \) at optimality, as estimated through some relaxed problem such as (20) or (23). Also, it is desirable to choose \( \theta_p \) values corresponding to the current incumbent solution.
COMPUTATIONAL RESULTS

Goal of Computational Comparisons

The main objective of the computational comparison is to develop the most effective branch and bound routine from the strategies presented, and to determine the conditions under which one bound may be preferable to another. Each of the lower bounding strategies described previously was coded in FORTRAN and run in an interactive mode on an IBM 3081 Model K computer in order to test its efficiency and overall performance in practice.

Problems were generated with the idea that they should be as representative as possible of an applied situation. Upper bounds somewhat less than the feasibility restriction $\frac{B}{C}$ were imposed on each variable. Computationally, this increased the difficulty of solving the integer knapsack problems and usually eliminated solutions with a single non-zero variable. The facility costs were generated according to a uniform distribution with a constant range and mean, and the amount of resource available was taken to be a fixed multiple of the maximum facility cost in each problem. The covering probabilities were also generated according to a uniform distribution, with selected values for the range and mean of the probabilities.

The sizes of problems solved was dependent in some cases on the maximum storage space available for keeping track of active nodes, as will be discussed later in this chapter. In general, the
problems used for testing the bounding strategies were small to medium-sized, and it is expected that problems of similar sized would be encountered in most applications.

Initial Comparison of Bounding Techniques

Each of the four bounding strategies presented in the foregoing discussion was tested computationally to assess its relative performance in solving a selected set of test problems. Both the integer and relaxed versions of problems (18) and (22) were included in the comparison, and the exact line search method and the inexact method were tested as separate cases of LB3. The performance characteristics of interest were the ability of each bounding technique to determine a strong lower bound on the objective function value at a given node, and its ability to provide a good incumbent solution via the use of some heuristic.

From trial runs, it was determined that characteristics of the customer-facility covering probabilities, in addition to problem size, significantly influenced the difficulty of the problem. In particular, the range of the probabilities and the mean of the probabilities for a problem seemed to determine the dominance of one bounding strategy over another. A set of range-mean combinations designed to test this observation is summarized in Table 1, and these combinations were used throughout the computational testing for problem generation.

Table 1. Probability Parameters

<table>
<thead>
<tr>
<th>Probability Parameters for $a_u$</th>
<th>Means for each range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Means for each range</td>
</tr>
<tr>
<td>0.10</td>
<td>0.95, 0.50, 0.05</td>
</tr>
<tr>
<td>0.30</td>
<td>0.85, 0.50, 0.15</td>
</tr>
<tr>
<td>0.50</td>
<td>0.75, 0.50, 0.25</td>
</tr>
<tr>
<td>0.70</td>
<td>0.65, 0.35</td>
</tr>
<tr>
<td>0.90</td>
<td>0.50</td>
</tr>
</tbody>
</table>

In order to compare only the bounding strategies without the influence of branching variable selection rules, lower bounds were calculated for node zero only for a set of 60 test problems. The bound value, the incumbent solution value, and the gap between each bound and the best
incumbent solution were recorded, along with the CPU time spent in determining each bound and incumbent solution. The strategies were then ranked according to how many times they provided the best bound for the problem, and how often they generated the best incumbent solution.

Table 2 summarizes the number of times that each bounding strategy gave the strongest lower bound value. Also, since the bounds were quite close in some cases, the next column of Table 2 records the number of times that each bound value was at most 5% away from the value of the best bound. Bounds are said to tie whenever this difference is 0.1% or less. Of the 60 problems tested, 10 cases arose where the bound values were so close to zero that they were rounded to zero to avoid computational difficulties. Since this rounding influences the discrimination between bounding strategies, these problems were excluded from the results shown in Table 2. Finally, the CPU time in seconds is recorded for the calculation of each bound for the set of 60 problems.

Table 2. Strength of Bounding Strategies

<table>
<thead>
<tr>
<th>Bound Technique</th>
<th># Times Best</th>
<th>Within 5% of Best</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer LB1</td>
<td>12</td>
<td>6</td>
<td>1.78</td>
</tr>
<tr>
<td>Continuous LB1</td>
<td>0</td>
<td>7</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Integer LB2</td>
<td>5</td>
<td>4</td>
<td>9.48</td>
</tr>
<tr>
<td>Continuous LB2</td>
<td>0</td>
<td>2</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>LB3 - exact search</td>
<td>26</td>
<td>13</td>
<td>1.27</td>
</tr>
<tr>
<td>LB3 - inexact search</td>
<td>23</td>
<td>18</td>
<td>1.08</td>
</tr>
<tr>
<td>LB4</td>
<td>4</td>
<td>6</td>
<td>3.4</td>
</tr>
</tbody>
</table>

The results summarized in Table 2 indicate that LB3 is by far the best technique for providing strong lower bounds for the node subproblem. Furthermore, the strength of LB3 is not significantly undermined by the use of the quadratic interpolation inexact line search method. The integer LB1 model gave the next best bounding results, but required somewhat more computation time. The set of knapsack problems that comprise integer LB2 required the most CPU time of all techniques tested and failed to give competitive bounding results. LB4 similarly required a large amount of CPU time and did not produce good bound values. The continuous versions of LB1 and LB2, while quick, also produced weak lower bound results.
Table 3 shows the relative capability of each bounding technique to produce a good incumbent solution value at node zero. A number of ties occurred here, especially among the smaller problems, and the eight cases where all incumbent solutions were equal have been excluded. The ability of LB4 to determine incumbent solutions was not included in this comparison.

Table 3. Incumbent Solution Strategies

<table>
<thead>
<tr>
<th>Incumbent Strategy</th>
<th>Best Upper Bound</th>
<th>Next Best Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Integer LB1</td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Continuous LB1</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Integer LB2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Integer LB2</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Continuous LB2</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Continuous LB2</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. LB3 - Exact Search</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. LB3 - Inexact Search</td>
<td>19</td>
<td>14</td>
</tr>
<tr>
<td>&amp; Heuristic Rule 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The two heuristic techniques used to round continuous solutions to integer solutions were both extremely quick, with negligible CPU time for the 60 test problems. However, the results indicate that a repeated solution of the integer knapsack problem comprising Heuristic 2 would require an appreciable amount of CPU time when a large number of nodes is enumerated.

As shown by the results in Table 3, Heuristic 1, which was tested with all the bounding procedures, gave the best results when used with the integer version of LB1. Both cases of LB3 followed closely. The continuous version of LB1, however, did not perform well, and both the integer and continuous versions of LB2, when combined with Heuristic 1, gave extremely poor results. Heuristic 2, tested with both cases of LB2, performed moderately well, but solving the integer knapsack problem for Heuristic 2 requires significantly more CPU time than for Heuristic 1.

Based on the overall strong performances of the Integer LB1 and the LB3 bounding techniques, we decided to develop these two strategies into full branch and bound routines for further computational results.
comparison. The Integer LB1 algorithm solves the problem (18) at each node to derive a lower bound, except at node zero, where both LB1 and LB3 are solved in order to hopefully obtain a good starting solution and a strong initial bound. The branching variable selection rule is based on the continuous relaxation of (18) which is also being solved at each node before computing LB1. The fractional variable that it produces (note that at most one variable is fractional) is selected for branching. This particular continuous bound was chosen because it is extremely quick and has the capability to provide a good incumbent solution.

At each node, then, the continuous relaxation of (18) is solved. If this bound fathoms the node, then another active node is selected for investigation, if one exists. If the continuous solution is integer-valued, then a priority ordering of the decision variables is used to select the next branching variable. If the continuous bound does not fathom the node, then the integer problem (18) is solved, and the bound LB1 is computed. In case this bound also does not fathom the node, the integer solution obtained is used as a candidate for updating the incumbent.

The branch and bound routine for LB3 is developed in a manner similar to that of the Integer LB1 algorithm. Again, LB1 and LB3 are both computed at node zero, and the continuous LB1 bound is obtained at each node in the hope of avoiding the more lengthy LB3 computation. If the continuous LB1 does not fathom the node, then LB3 is calculated and, if possible, the node is fathomed.

Several choices for a branching variable selection rule were tested on a small set of problems. Following the work of Sherali and Myers (1984), the following selection rules were compared: 1) fractionality of the variables, and 2) fractionality times gradient component. The second rule, variable fractionality times gradient component, performed best in the problems tested, with the first rule, fractionality alone, ranking a close second. Since the gradient information is readily available after computing LB3, and since these results are similar to the more extensive findings of Sherali and Myers, we decided to use the fractionality times gradient rule for branching variable selection in the LB3 algorithm.

The computational efficiency of the LB3 calculation was a concern in the early phases of the comparisons. In order to reduce the time required to solve problems with LB3, several parameters
were adjusted. First, the number of line search iterations performed at each node has a major effect on overall time required. Several problems were tested where the number of line search iterations was limited to 2, 4, 6, and 8 iterations at each node. Naturally, as the number of iterations decreases, the bound is potentially weaker, and the number of nodes enumerated is expected to increase.

The problems tested indicated that the increase in nodes enumerated is not significant with a limit of 4 or 6 line search iterations, but with only 2 iterations per node, a sizeable increase did occur. For this reason, the limit on the maximum line search iterations per node was set at 4 throughout the subsequent comparisons of the algorithms. Additionally, the inexact line search procedure terminates when an uncertainty of 10% or less is attained over the specified interval. Again, this factor was determined by balancing the tradeoff between potentially stronger bounds and decreased computation time.

For purposes of comparison, a third branch and bound algorithm was developed in which only the continuous version of LB1 was used as a bounding strategy. This bound, while weaker than the integer LB1 and the LB3 bounds, is extremely easy to compute, and was thought to be potentially competitive with the other two strategies with respect to the total CPU time required.

Evaluation of Algorithms

An initial set of 9 problems was solved using all three algorithms. The parameters of the problems in this comparison were chosen as a representative subset of the values in Table 1. The detailed outputs from this comparison are contained in Appendix B, while Table 4 summarizes these results. The CPU times for these results as well as those in the following tables are in seconds.
Table 4. Results of Problem Set

<table>
<thead>
<tr>
<th>Size (Range-Mean)</th>
<th>Cont. LB1</th>
<th>Int. LB1</th>
<th>LB3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>CPU</td>
<td>Nodes</td>
</tr>
<tr>
<td></td>
<td>Enumerator</td>
<td></td>
<td>Enumerator</td>
</tr>
<tr>
<td>15x10 (.1-.50)</td>
<td>1499</td>
<td>2.2</td>
<td>527</td>
</tr>
<tr>
<td>15x10 (.3-.50)</td>
<td>721</td>
<td>1.0</td>
<td>301</td>
</tr>
<tr>
<td>20x12 (.5-.75)</td>
<td>3455</td>
<td>7.7</td>
<td>2123</td>
</tr>
<tr>
<td>15x10 (.5-.50)</td>
<td>1827</td>
<td>3.1</td>
<td>1179</td>
</tr>
<tr>
<td>20x12 (.5-.25)</td>
<td>8259</td>
<td>20.5</td>
<td>4447</td>
</tr>
<tr>
<td>15x10 (.7-.65)</td>
<td>3663</td>
<td>6.7</td>
<td>1869</td>
</tr>
<tr>
<td>10x10 (.7-.35)</td>
<td>5021</td>
<td>7.4</td>
<td>3091</td>
</tr>
<tr>
<td>20x12 (.7-.35)</td>
<td>15137</td>
<td>48.5</td>
<td>12621</td>
</tr>
<tr>
<td>20x12 (.9-.50)</td>
<td>14263</td>
<td>46.5</td>
<td>10737</td>
</tr>
</tbody>
</table>

These results indicate that LB3 is the most efficient algorithm with respect to the number of nodes enumerated, especially as the \( \sigma_y \) range increases. This efficiency in enumeration, however, does not always translate to computational savings. In fact, LB3 is the fastest algorithm in only 2 of the 9 cases shown in Table 4.

A look at these results indicates that while the continuous version of LB1 requires the greatest number of nodes in the enumeration tree, it is computationally so much quicker than the other two strategies that it is almost always the dominant strategy with respect to CPU time. Between the other two strategies, the LB3 branch and bound routine usually enumerates fewer nodes than the integer LB1 routine, but there exists no clear dominance between the strategies with respect to CPU time.

The main problem with the continuous LB1 bound is that it enumerates so many nodes that storage space may quickly become a problem. Additionally, for some larger problems, the continuous bounds are too weak to solve the problem in a reasonable amount of time. For these reasons, a hybrid algorithm that relies primarily on the continuous bound but also computes one of the stronger bounds at appropriate nodes has been developed. To assess the value of computing either integer LB1 or LB3 at a given node, the gap between the incumbent solution and the continuous LB1 solution was measured at those nodes where the continuous bound did not fathom the node but the integer LB1 bounds or the LB3 bound did fathom. This average gap is recorded for each problem in the preceding tables. The purpose of this measurement is to devise a means of judging...
at each node the likelihood that the computation of a stronger bound will fathom the node. If it seems likely, based on this test, that the extra computational effort will fathom the node, then the second bound will be computed. Otherwise, the continuous LB1 will be the only bounding strategy employed at the node.

A second set of 9 problems was tested in order to further observe the performance of the three algorithms. In this set, the problem size has been increased and the number of customers is several times larger than the number of available facilities. The other problem parameters are similar to those in the previous set of problems. Detailed results of these problems are contained in Appendix B, and Table 5 summarizes the key performance measures.

### Table 5. Results of Problem Set II

<table>
<thead>
<tr>
<th>Size (Range-Mean)</th>
<th>Cont. LB1</th>
<th></th>
<th>Int. LB1</th>
<th></th>
<th>LB3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>CPU</td>
<td>Nodes</td>
<td>CPU</td>
<td>Nodes</td>
</tr>
<tr>
<td></td>
<td>Enumerator</td>
<td></td>
<td>Enumerator</td>
<td></td>
<td>Enumerator</td>
</tr>
<tr>
<td>50x10 (.1-.50)</td>
<td>1541</td>
<td>5.8</td>
<td>661</td>
<td>4.5</td>
<td>1019</td>
</tr>
<tr>
<td>30x10 (.3-.50)</td>
<td>1585</td>
<td>4.1</td>
<td>869</td>
<td>4.7</td>
<td>445</td>
</tr>
<tr>
<td>40x12 (.5-.75)</td>
<td>3187</td>
<td>12.6</td>
<td>2321</td>
<td>14.2</td>
<td>123</td>
</tr>
<tr>
<td>50x10 (.5-.50)</td>
<td>2381</td>
<td>10.3</td>
<td>1515</td>
<td>11.6</td>
<td>349</td>
</tr>
<tr>
<td>30x12 (.5-.25)</td>
<td>12319</td>
<td>43.8</td>
<td>6435</td>
<td>40.8</td>
<td>791</td>
</tr>
<tr>
<td>40x10 (.7-.65)</td>
<td>2553</td>
<td>9.4</td>
<td>1539</td>
<td>10.3</td>
<td>159</td>
</tr>
<tr>
<td>50x10 (.7-.35)</td>
<td>7745</td>
<td>39.0</td>
<td>7375</td>
<td>47.4</td>
<td>219</td>
</tr>
<tr>
<td>30x12 (.7-.35)</td>
<td>16707</td>
<td>63.9</td>
<td>14013</td>
<td>77.9</td>
<td>133</td>
</tr>
<tr>
<td>40x12 (.9-.50)</td>
<td>16409</td>
<td>76.3</td>
<td>13041</td>
<td>87.2</td>
<td>345</td>
</tr>
</tbody>
</table>

Again, these results indicate a very good performance by LB3 in terms of enumerating a minimal tree. The notable exception to this rule occurs when the \(a_y\) range is small; in this case, as expected, LB1 is dominant. Performance with respect to CPU time is fairly evenly split between the algorithms - integer LB1 is fastest three times, continuous LB1 - four times, and LB3 has two runs with the best CPU time.

A third set of problems was tested in which the number of available facilities is larger than the number of customers. It was expected, based on the previous runs and the characteristics of the problems where LB3 was better than LB1, that LB3 would have a strong performance in these problems. Table 6 gives a summary of these results, and the detailed outputs are found in Appendix B.
Table 6. Results of Problem Set III

<table>
<thead>
<tr>
<th>Size (Range-Mean)</th>
<th>Cont. LB1</th>
<th>Int. LB1</th>
<th>LB3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes Enumer.</td>
<td>CPU</td>
<td>Nodes Enumer.</td>
</tr>
<tr>
<td>8x20 (.1-.50)</td>
<td>20117</td>
<td>45.8</td>
<td>16709</td>
</tr>
<tr>
<td>7x25 (.3-.50)</td>
<td>1567</td>
<td>2.9</td>
<td>1367</td>
</tr>
<tr>
<td>10x15 (.5-.75)</td>
<td>6181</td>
<td>12.7</td>
<td>3931</td>
</tr>
<tr>
<td>8x20 (.5-.50)</td>
<td>10009</td>
<td>22.2</td>
<td>8143</td>
</tr>
<tr>
<td>7x25 (.5-.25)</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>10x15 (.7-.65)</td>
<td>3193</td>
<td>5.8</td>
<td>2163</td>
</tr>
<tr>
<td>5x10 (.7-.35)</td>
<td>3745</td>
<td>4.8</td>
<td>2571</td>
</tr>
<tr>
<td>7x25 (.7-.35)</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>5x10 (.9-.50)</td>
<td>2013</td>
<td>2.2</td>
<td>1215</td>
</tr>
</tbody>
</table>

* indicates that available storage space is exceeded

Table 6 shows that, as expected, LB3 is dominant for most of these problems with respect to both number of nodes enumerated and CPU time required. Another notable advantage of the LB3 routine is that it was able to solve problem number 8, which was not solved by either the integer LB1 or the continuous LB1 algorithms. Storage space on all three routines was limited to 2200 active nodes stored simultaneously.

As a result of the problems encountered in the storage of excessive numbers of active nodes in the larger problems, a depth-first approach was compared with the original pure breadth-first routine. The results of the depth-first enumerations are summarized in Table 7, with the detailed output given in Appendix B.
Table 7. Results of Problem Set II - Depth-First Enumeration

<table>
<thead>
<tr>
<th>Size (Range-Mean)</th>
<th>Nodes Enumer.</th>
<th>CPU</th>
<th>Nodes Enumer.</th>
<th>CPU</th>
<th>Nodes Enumer.</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>50x10 (.1-.50)</td>
<td>2915</td>
<td>6.2</td>
<td>777</td>
<td>4.7</td>
<td>1907</td>
<td>43.0</td>
</tr>
<tr>
<td>30x10 (.3-.50)</td>
<td>2917</td>
<td>4.2</td>
<td>1125</td>
<td>5.8</td>
<td>1267</td>
<td>21.7</td>
</tr>
<tr>
<td>40x12 (.5-.75)</td>
<td>5797</td>
<td>12.4</td>
<td>2123</td>
<td>13.5</td>
<td>233</td>
<td>17.7</td>
</tr>
<tr>
<td>50x10 (.5-.50)</td>
<td>4321</td>
<td>10.4</td>
<td>1515</td>
<td>11.5</td>
<td>1025</td>
<td>42.6</td>
</tr>
<tr>
<td>30x10 (.5-.25)</td>
<td>22307</td>
<td>39.1</td>
<td>14101</td>
<td>75.0</td>
<td>1461</td>
<td>50.9</td>
</tr>
<tr>
<td>40x10 (.7-.65)</td>
<td>4673</td>
<td>9.5</td>
<td>1577</td>
<td>10.2</td>
<td>673</td>
<td>31.9</td>
</tr>
<tr>
<td>50x10 (.7-.35)</td>
<td>13847</td>
<td>36.7</td>
<td>7395</td>
<td>43.3</td>
<td>537</td>
<td>51.3</td>
</tr>
<tr>
<td>30x12 (.7-.35)</td>
<td>30241</td>
<td>56.5</td>
<td>19929</td>
<td>94.1</td>
<td>425</td>
<td>30.4</td>
</tr>
<tr>
<td>40x12 (.9-.50)</td>
<td>29937</td>
<td>68.9</td>
<td>21505</td>
<td>130.9</td>
<td>865</td>
<td>101.9</td>
</tr>
</tbody>
</table>

Comparison with Table 5 shows that there was no improvement with respect to CPU time, while the number of nodes enumerated either increased or remained the same.

An additional computational test was performed to see whether a large number of nodes were being enumerated as a result of bounds that were extremely close to fathoming. The tolerance percentage for fathoming nodes was set at 5% for this comparison, rather than the 0.1% that was used in all previous computations. Table 8 gives the results for the increased fathoming percentage for Problem Set II, with the detailed results in Appendix B. As shown by this comparison, the increased fathoming did not significantly affect the solution values, and the savings in CPU time and numbers of nodes enumerated was substantial.

Table 8. Results of Problem Set II - Increased Fathoming Tolerance

<table>
<thead>
<tr>
<th>Size (Range-Mean)</th>
<th>Nodes Enumer.</th>
<th>CPU</th>
<th>Nodes Enumer.</th>
<th>CPU</th>
<th>Nodes Enumer.</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>50x10 (.1-.50)</td>
<td>1483</td>
<td>5.3</td>
<td>393</td>
<td>2.9</td>
<td>991</td>
<td>37.3</td>
</tr>
<tr>
<td>30x10 (.3-.50)</td>
<td>1301</td>
<td>3.2</td>
<td>471</td>
<td>3.0</td>
<td>281</td>
<td>8.8</td>
</tr>
<tr>
<td>40x12 (.5-.75)</td>
<td>1</td>
<td>0.7</td>
<td>1</td>
<td>0.7</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>50x10 (.5-.50)</td>
<td>2311</td>
<td>10.0</td>
<td>1515</td>
<td>11.4</td>
<td>339</td>
<td>31.0</td>
</tr>
<tr>
<td>30x12 (.5-.25)</td>
<td>12191</td>
<td>43.2</td>
<td>6383</td>
<td>40.0</td>
<td>785</td>
<td>50.8</td>
</tr>
<tr>
<td>40x10 (.7-.65)</td>
<td>2265</td>
<td>8.3</td>
<td>1511</td>
<td>9.7</td>
<td>45</td>
<td>5.5</td>
</tr>
<tr>
<td>50x10 (.7-.35)</td>
<td>7709</td>
<td>38.8</td>
<td>7287</td>
<td>46.4</td>
<td>123</td>
<td>27.8</td>
</tr>
<tr>
<td>30x12 (.7-.35)</td>
<td>16415</td>
<td>62.6</td>
<td>13399</td>
<td>74.3</td>
<td>91</td>
<td>16.7</td>
</tr>
<tr>
<td>40x12 (.9-.50)</td>
<td>15941</td>
<td>74.0</td>
<td>12303</td>
<td>81.8</td>
<td>191</td>
<td>53.2</td>
</tr>
</tbody>
</table>
Finally, a hybrid strategy was developed in which the Continuous LB1 was used as the primary branching strategy, with LB3 computed at certain stages of the enumeration tree. This strategy was tested in hopes of having the LB3 bound fathom large sections of the tree whenever it is computed, and letting the weaker Continuous LB1 bound operate as usual otherwise. Several criteria were devised to determine when LB3 should be computed. The rules that were tested included the following:

1. If the gap between a Continuous LB1 bound value that does not fathom the node and the incumbent solution value at that node is less than a certain percentage, compute LB3.

2. If the level of the tree is greater than some prescribed number and criterion 1 is met, compute LB3.

3. If the level of the tree is equal to a multiple of some prescribed number, compute LB3.

4. Criteria 2 and 3 together.

It was decided to use the hybrid approach within a depth-first routine since the potential savings from early fathoming is greater. Criteria 3 performed better than the other three rules tested, and the results of Problem Set I solved with the hybrid algorithm using this rule are presented in Table 9, along with the results from the depth-first routine for comparison.

**Table 9. Results of Problem Set I - Hybrid and Depth-First Algorithms**

<table>
<thead>
<tr>
<th>Size (Range-Mean)</th>
<th>Hybrid Algorithm</th>
<th>Pure Depth-First</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes Enumerator</td>
<td>CPU</td>
</tr>
<tr>
<td>15x10 (.1-.50)</td>
<td>1479</td>
<td>2.6</td>
</tr>
<tr>
<td>15x10 (.3-.50)</td>
<td>949</td>
<td>1.8</td>
</tr>
<tr>
<td>20x12 (.5-.75)</td>
<td>3373</td>
<td>11.2</td>
</tr>
<tr>
<td>15x10 (.5-.50)</td>
<td>1827</td>
<td>3.5</td>
</tr>
<tr>
<td>20x12 (.5-.25)</td>
<td>8683</td>
<td>19.0</td>
</tr>
<tr>
<td>15x10 (.7-.65)</td>
<td>3999</td>
<td>6.2</td>
</tr>
<tr>
<td>10x10 (.7-.35)</td>
<td>6555</td>
<td>7.8</td>
</tr>
<tr>
<td>20x12 (.7-.35)</td>
<td>15119</td>
<td>32.4</td>
</tr>
<tr>
<td>20x12 (.9-.50)</td>
<td>14311</td>
<td>31.8</td>
</tr>
</tbody>
</table>
By comparing the results of the hybrid algorithm with those of the pure depth-first routine, it can be seen that, overall, the hybrid algorithm did not perform as well as had been hoped, since the CPU time never decreased below the times recorded for the regular depth-first strategy. However, the savings in nodes enumerated is substantial in some cases, and may be useful when storage space is restricted. By varying parameters such as the prescribed level beyond which LB3 should be computed, or the size of the bound-versus-incumbent gap that triggers an LB3 computation, a balance can be determined between the number of nodes enumerated and the CPU time required.
CONCLUSIONS

In this thesis, two generalized set covering formulations have been presented. The usual formulations of the total and partial set covering models have been shown to be special cases of the general models, and some applications of the general models have been mentioned. Additionally, solution strategies for the generalized formulations have been discussed, and algorithms for a specialized case of the partial covering model have been developed.

The computational results of these algorithms indicate that the continuous LB1 routine performs consistently better than either the Integer LB1 or the LB3 routines with respect to CPU time. When the range of the covering probabilities is small, LB1 is especially strong, and the theoretical basis for this result has been discussed.

It has been pointed out that storage space limitations become important in those cases where a large number of nodes must be stored simultaneously, and LB3 has demonstrated the ability to enumerate a much smaller tree than either of the other routines. Therefore, in solving very large problems, it may be desirable to use the LB3 routine in spite of its longer computation time.
Appendix A. Listing of FORTRAN Code

CSJOB  PARRISH,NOLIST

C

C MAIN PROGRAM CALLS INITIALIZATION ROUTINE, PROBLEM GENERATION
C ROUTINE, AND MAIN DRIVER OF OPTIMIZATION PROCEDURE. WHEN PROBLEM
C IS SOLVED, MAIN PROGRAM PRINTS PROBLEM STATISTICS.
C

COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /INIT/ PROB(50,50)
COMMON /PRAM/ RANG,XMID,CRAN,CMID
COMMON /KGAP/ KNT,AVGAP
COMMON /KGP/ XGAP
COMMON /KNTR/ NFATH(4),KOUNTN,FATH,NLIST,NINCSL
COMMON /XLBD/ ZSTAR,KKLB,ORGINC
DIMENSION XBISOL(50)
REAL LB1, LB3
EPS = 0.0001
ITER = 9
FATH = 0.001
WRITE(6,106)
WRITE(6,101)
DO 55 KITER = 1, ITER
CALL TIMEON
CALL GENP
CALL ILZE
CALL TREE

101 FORMAT(1x,'INTEGER LB1 - 10 KNAPSACK INTERATIONS ',//,95(' ','/)
CALL TIMECK(NTIME)
TM = NTIME/100.
XIMP = 0.
PFATH = 0.
PN1F = 0.
PN2F = 0.
IF(KNT.GT.0)AVGAP = AVGAP/FLOAT(KNT)
TFTH = 0.
DO 86 KL = 1,4
TFTH = TFTH + NFATH(KL)
86 CONTINUE
IF(TFTH.LT.EPS)GO TO 356
PFATH = TFTH/FLOAT(KOUNTN)
PN1F = NFATH(1)/TFTH
PN2F = NFATH(2)/TFTH
356 IF(ORGINC.GT.EPS)XIMP = (ORGINC-ZSTAR)/ORGINC*100.
106 FORMAT(1X,7X,'PROB','10X','PROB NODES NODES OPT OPT ','
*AVG TOTAL CLB1 CLB3 '/
*1X,'NO. RANGE SIZE MEAN ENUM STOR SOLN NODE ','
*GAP FATH FATH FATH CPU'!)
WRITE(6,107)KITER,RANG,NCUS,NFAC,XMID,KOUNTN,NLIST,ZSTAR,
* NINCSL,AVGAP,PFATH,PN1F,PN2F,TFTH
107 FORMAT(1X,I3,2X,F5.2,2X,I2,12X,'X','J2,2X,F5.2,2(2X,I7),2X,E9.4,
* 2X,I4,2X,3(F6.2,2X),F5.2,2X,F7.3,/')
55 CONTINUE
STOP
END

SUBROUTINE ILZE

C THIS SUBROUTINE INITIALIZES SEVERAL VARIABLES AND ARRAYS AT THE
C START OF THE SOLUTION PROCEDURE

C

COMMON /NLST/ XKMAX(50)
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /INIT/ PROB(50,50)
COMMON /LAMB/ RAMBJ(50),RAMB(50,50),RATX(50)
COMMON /KPSK/ IRJ(50)

C

C CALCULATE INITIAL LAMBDA VALUES AND RANK ORDERING OF VARIABLES

C

DO 11 J = 1,NFAC
    RA = 0.
    DO 10 I = 1,NCUS
        RAMBJ(I,J) = -ALOG(PROB(I,J))
        RA = RA + RAMBJ(I,J)
    10 CONTINUE
    RAMBJ(J) = RA
11 CONTINUE

DO 12 J = 1,NFAC
    RATX(J) = RAMBJ(J)/COST(J)
    IRJ(J) = J
12 CONTINUE

NF1 = NFAC-1
DO 13 J = 1,NF1
    JP1 = J + 1
    DO 13 J1 = JP1,NFAC
        IF(RATX(J).GT.RATX(J1))GO TO 13
        IA = IRJ(J)
        IRJ(J) = IRJ(J1)
        IRJ(J1) = IA
13 CONTINUE
RETURN
END

C
C

Appendix A. Listing of FORTRAN Code
SUBROUTINE TREE

C
C THIS IS THE MAIN DRIVER OF THE OPTIMIZATION ROUTINE. THE TREE IS
C ENUMERATED WITHIN THIS SEGMENT, AND TESTING FOR OPTIMALITY IS
C PERFORMED. IF OPTIMALITY IS NOT REACHED, A NODE IS SELECTED FOR
C FURTHER EXPLORATION.
C
C REAL LB1
COMMON /DLST/ KNODE(1100,4),XLBND(1100),TBND(1100,50),
PSOL(1100,50)
COMMON /NLST/ XMMAX(50)
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /BNDs/ XLB,LB1,LB3,KNBND
COMMON /KNTR/ NFATH(4),KOUNTN,FATH,NLIST,NINCSL
COMMON /KPGP/ XGAP
COMMON /KGAP/ KNT,AVGAP
COMMON /XLBD/ ZSTAR,KKL,B,ORGINC
COMMON /CLBN/ CLBN1(50)
COMMON /ITER/ KBRN,IBVAL,IK,IK2,NF2,CRHS,IP,IBP
COMMON /CHKI/ KINC
COMMON /ZRON/ KNZRO,NZRO(50)
COMMON /SHRC/ CSTAR
DIMENSION XREMN(50),XLB(50),XBISOL(50),XBOL1(50),XBOL2(50)
NINCSL = 0
EPS = 0.00001
C
C INITIALIZED OF SCALAR AND ARRAY VARIABLES
C
DO 14 J = 1,NFAC
XLBN(J) = 0.
14 CONTINUE
KOUNTN = 1
NLIST = 1
KNT = 0
AVGAP = 0.
CSTAR = 99898.
DO 114 KK = 1,4
  NFATH(KK) = 0
114 CONTINUE
  DO 141 IT = 1,1100
  DO 142 IT2 = 1,4
    KNODE(IT,IT2) = 0
142 CONTINUE
  XLBNDD(IT) = 0.
  DO 144 IT4 = 1,50
    TBND(IT,IT4) = 0.
    PSOL(IT,IT4) = 0.
144 CONTINUE
141 CONTINUE
  ZSTAR = 998.
  KKL = 0.
C
C TEST FOR VARIABLES TO ELIMINATE
C
  KNZRO = 0
  DO 10 J = 1,NFAC
    IF(ABS(XKMAX(J)-XLBN(J)).LT.EPS) GO TO 10
    IF(XKMAX(J).LT.XLBN(J)) GO TO 11
    KNZRO = KNZRO + 1
    NZRO(KNZRO) = J
10 CONTINUE
  GO TO 100
11 WRITE(6,12)
12 FORMAT(' SOLUTION JS INFEASIBLE AT NODE ZERO ')
RETURN
C
C OBTAIN INITIAL INCUMBENT SOLUTION AND STORE LBISOL IN LIST
C
100 CRHS = RHS
  CALL ZLBI(XLB1,XLBN,XKMAX,XBSOL1,VAL1)
  CALL CLBI(XLB2,XLBN,XKMAX,XBSOL2,VAL2)
  XLB = AMAX1(XLB1,XLB2)
IF(VAL1.GT.VAL2)GO TO 78
DO 23 J = 1,NFAC
XBISOL(J) = XBSOL1(J)
23 CONTINUE
GO TO 25
78 DO 24 J = 1,NFAC
XBISOL(J) = XBSOL2(J)
24 CONTINUE
25 LBI = XLB
C
C CHECK FOR OPTIMALITY
C
CALL INCUM(I,CSTAR,XBISOL)
IF(XLB.GT.CSTAR) RETURN
XGAP = 0.
IF(ZSTAR.GT.EPS)XGAP = ((ZSTAR-XLB)/CSTAR)*100.
CALL BRAN(1,XLB,N,KERR,XREMN,KINTGR,XBISOL)
IF(KERR.EQ.1) RETURN
KNOE(NLIST,1) = 1
KNOE(NLIST,2) = 0
KNOE(NLIST,3) = 1
KNOE(NLIST,4) = FLOAT(KBRN * 100) + Int(CLBN1(KBRN))
XLBND(NLIST) = XLB
DO 16 J = 1,NFAC
TBND(I,J) = XKMAX(J) + XLBN(J)*100
PSOL(I,J) = XBISOL(J)
16 CONTINUE
C
C CHOOSE NODE WITH LEAST LOWER BOUND
C
999 SLB = 999.
IBP = 0
NACT = MINO(1100,NLIST)
DO 21 I = 1,NACT
IF(KNODE(I,1).EQ.0) GO TO 21
IF(XLBND(I,GT.SLB) GO TO 21
IBP = 1
SLB = XLBND(I)
21 CONTINUE
C
C CHECK OPTIMALITY
C
IF(IBP.EQ.0) RETURN
C
C OBTAIN BRANCHING VARIABLE AND VALUE FROM LIST (IBP,2)
C
IK = KNODE(IBP,4)/100
IBVAL = KNODE(IBP,4)-IK*100
C
C CREATE A NEW NODE ON LOWER SIDE OF BRANCHING VARIABLE
C
CRHS = RHS
KOUNTN = KOUNTN + 1
DO 22 J = 1,NFAC
NVAL = TBND(IBP,J)
XBISOL(J) = PSOL(IBP,J)
XLBND(J) = NVAL/100
XKMAX(J) = NVAL-INT(XLBND(J)*100)
XREM(J) = XKMAX(J)-XLBND(J)
IF(XLBND(J),GT,EPS)CRHS = CRHS - COST(J) * XLBND(J)
IF(J.NE.IK)GO TO 22
XKMAX(J) = FLOAT(IBVAL)
XREM(J) = XKMAX(J)-XLBND(J)
IF(XREM(J),GT,E0)GO TO 22
NFATH(J) = NFATH(J) + 1
GO TO 345
22 CONTINUE
C
C TEST FOR VARIABLES TO ELIMINATE
C
KNZRO = 0
DO 20 J = 1,NFAC
IF(ABS(XREMN(J)).LT.0.1)GO TO 20
KNZRO = KNZRO + 1
NZRO(KNZRO) = J
20 CONTINUE
IF(KNZRO.NE.0)GO TO 103
NFATH(2) = NFATH(2) + 1
GO TO 345

C CHECK WHETHER PREVIOUS LBISOL IS FEASIBLE IN CURRENT NODE OR NOT
C  (IND1 = 1 IF FEASIBLE)
C
103 IND1 = 2
IF(IBVALGE.INT(XBISOL(IK))) IND1 = 1
CALL ILBI(IND1,XLB,XLBN,XREMN,XBISOL,IELP,BIGAP)
IF(XLB.LT.CSTAR)GO TO 245
IF(IELP.EQ.1)GO TO 746
NFATH(1) = NFATH(1) + 1
GO TO 345
746 NFATH(2) = NFATH(2) + 1
AVGAP = AVGAP + BIGAP
KNT = KNT + 1
GO TO 345
245 CALL INCUM(2,CSTAR,XBISOL)
    CALL BRAN(2,XLBN,KERR,XREMN,KINTGR,XBISOL)
    IF(KERR.EQ.1)RETURN
C
C CREATE NEW NODE ON UPPER SIDE OF BRANCHING VARIABLE
C
345 CRHS = RHS
    KOUNTN = KOUNTN + 1
    DO 181 J = 1,NFAC
    J2 = J + 2
    NVAL = TBND(IBP,J)
    XBISOL(J) = PSOL(IBP,J)
    XLBN(J) = NVAL/100
    XKMAX(J) = NVAL-INT(XLBN(J)*100)
XREMN(J) = XKMAX(J)-XLBN(J)
IF(J.NE.IK)GO TO 182
XLBN(J) = FLOAT(IBVAL)+1.
XREMN(J) = XKMAX(J)-XLBN(J)
IF(XREMN(J).LT.O.)GO TO 400
182 IF(XLBN(J).GT.EPS)CRHS = CRHS-COST(J)*XLBN(J)
   IF(CRHS.LE.O.)GO TO 400
181 CONTINUE
   GO TO 202
C
C BRANCH IS INFEASIBLE
C
400 XLBND(IBP) = 9999.
   NFATH(3) = NFATH(3)+1
   GO TO 423
C
C TEST FOR VARIABLES TO ELIMINATE
C
202 KNZRO = 0
   DO 30 J = 1,NFAC
      IF(ABS(XREMN(J)).LT.O.l)GO TO 30
      KNZRO = KNZRO +1
      NZRO(KNZRO) = J
30 CONTINUE
   IF(KNZRO.NE.0)GO TO 388
   NFATH(2) = NFATH(2)+1
   GO TO 423
388 IND1 = 2
   IF((IBVAL + 1).LE.INT(XBISOL(IK)))IND1 = 1
   CALL ILB1(IND1,XLB,XLBN,XREMN,XBISOL,IELP,BIGAP)
   IF(XLB.LT.CSTAR)GO TO 346
   IF(IELP.EQ.1)GO TO 747
   NFATH(1) = NFATH(1)+1
   GO TO 423
747 NFATH(2) = NFATH(2)+1
   AVGAP = AVGAP + BIGAP
KNT = KNT + 1
GO TO 423

346 CALL INCUM(2,CSTAR,XBISOL)
   CALL BRAN(2,XLBN,KERR,XREMN,KINTGR,XBISOL)
   IF(KERR.EQ.1)RETURN

423   KNODE(IBP,1) = 0
   GO TO 999
   END

C
C

SUBROUTINE ILBI(IND,LBl,XLBN,XREMN,XBISOL,IELP,BIGAP)

C
C THIS SUBROUTINE FINDS THE INTEGER BOUND LBI, BASED ON THE
C ARITHMETIC VERSUS GEOMETRIC MEAN OF A SET OF TERMS.
C
C
REAL LBI
COMMON /DLST/ KNODE(1100,4),XLBN(1100),TBND(1100,50),
   PH(1100,50)
COMMON /XLBD/ ZSTAR,KKLB,ORGINC
COMMON /NLST/ XKMAX(50)
COMMON /LINE/ XMINL,DJ(50)
COMMON /ITER/ KBRN,IBVAL,IK,IK2,NF2,CRHS,IP,IBP
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /LAMB/ RAMB(50),RAMB(50,50),RATX(50)
COMMON /CLBN/ CLBN1(50)
COMMON /KPSK/ IRJ(50)
COMMON /MINC/ CJMIN
COMMON /KNTR/ NFATH(4),KOUNTN,FATH,NLIST,NINCSL
COMMON /SHRC/ CSTAR
DIMENSION ISOLB(50),RLB(50),XREMN(50),XLBN(50),CBISOL(50)
DIMENSION XBISOL(50)
EPS = 0.0001
XMINL = 0.
IELP = 0
C

Appendix A. Listing of FORTRAN Code  58
C FIND THE CONTINUOUS LB1 SOLUTION

C
XRH = CRHS
LB1 = 0.
DO 75 J = 1,NFAC
RLB1(J) = RAMBJ(J)/COST(J)
75 CONTINUE
CALL LINK(XRH,XREMN,RLB1,CLBN1)
DO 22 J = 1,NFAC
CLBN1(J) = CLBN1(J) + XLBN(J)
22 CONTINUE
IF(KOUNTN.EQ.1)GO TO 323

C SEE IF CONTINUOUS SOLUTION FATHOMS

C
20 CALL COMPBI(CLBN1,LB1)
IF(LB1.GE.CSTAR)RETURN
BIGAP = ((CSTAR-LB1)/CSTAR)*100.
IF(IND.EQ.1)GO TO 77

C IF BRANCHING HAS CUT OFF THE INTEGER SOLUTION, FIND A NEW INT. SOLN.

C
323 KKLB = KKLB + 1
IELP = 1
XRH = CRHS
POBJ = 0.
CALL KNAP(XRH,XREMN,RLB1,RAMBJ,ISOLB1,10,OBJK)
DO 23 J = 1,NFAC
XBISOL(J) = ISOLB1(J) + XLBN(J)
POBJ = POBJ + XLBN(J)*RAMBJ(J)
23 CONTINUE
XPON = (OBJKS + POBJ)*(-1.)*1/(1./NCUS)
LB1 = EXP(XPON)*NCUS
RETURN

C

Appendix A. Listing of FORTRAN Code 59
C FIND INCUMBENT BASED ON CONTINUOUS SOLUTION

C

77 DO 10 J = 1,NFAC
    XBISOL(J) = INT(CLBN1(J))
10 CONTINUE
XRH = CRHS
DO 1 J = 1,NFAC
   IF(XBISOL(J).LT.EPS) GO TO 1
   XRH = XRH - COST(J)*XBISOL(J)
   IF(XRH.LT.CJMIN) GO TO 4
1 CONTINUE
DO 2 KFAC = 1,NFAC
DO 3 J = 1,NFAC
   IF(IRJ(J).NE.KFAC) GO TO 3
   IF(COST(J).GT.XRH) GO TO 2
   MAXA = XKMAX(J)-CBISOL(J)
   MAXB = XRH/COST(J)
   IF(MAXB.LT.MAXA) MAXA = MAXB
   XBISOL(J) = XBISOL(J) + MAXA
   XRH = XRH - COST(J)*MAXA
   IF(XRH.LT.CJMIN) GO TO 4
3 CONTINUE
2 CONTINUE
4 RETURN
END

C

SUBROUTINE ZLB1(LBI,XLBN,XREMNXSOL,VALI)

C

C THIS SUBROUTINE COMPUTES THE INTEGER LB1 VALUE DURING THE FIRST
C ITERATION, AND FINDS THE OBJECTIVE FUNCTION VALUE AT THE ASSOCIATED
C SOLUTION.

C

REAL LBI
COMMON /DLST/ KNODE(1100,4),XLBN(1100),TBND(1100,50),
* PSOL(1100,50)
COMMON /XLBD/ ZSTAR,KKLB,ORGINC
COMMON /NLST/ XKMAX(50)
COMMON /LINE/ XMINL,DJ(50)
COMMON /ITER/ KBRN,IBVAL,IK,IK2,NF2,CRHS,IP,IBP
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /LAMB/ RAMBJ(50),RAMB(50,50),RATX(50)
COMMON /CLBN/ CLBN1(50)
COMMON /KPSK/ IRJ(50)
COMMON /MINC/ CJMIN
DIMENSION ISOLB(50),RLB1(50),XREMN(50),XLBN(50),CB1SOL(50)
DIMENSION XSOL(50)
EPS = 0.0001
XMINL = 0.
KKLB = KKLB + 1
XRH = CRHS
POBJ = 0.
DO 75 J = 1,NFAC
   RLB1(J) = RAMBJ(J)/COST(J)
75 CONTINUE
CALL KNAP(XRH,XREMN,RLB1,RAMBJ,ISOLB1,OBJKS)
DO 23 J = 1,NFAC
   XSOL(J) = ISOLB1(J) + XLBN(J)
   POBJ = POBJ + XLBN(J)*RAMBJ(J)
23 CONTINUE
XPON = (OBJKS + POBJ)*(-1.)*1./NCUS
LB1 = EXP(XPON)*NCUS
CALL COMPU(0.,VAL1,XSOL,0)
RETURN
END

C
C
SUBROUTINE LB2
C
C THIS ROUTINE CALCULATES THE SECOND LOWER BOUND, BASED ON SUM OF
C MINIMUM TERMS VERSUS MINIMUM OF SUM OF TERMS.(NOT CALLED IN THIS
C VERSION OF THE PROGRAM.)
COMMON /DLST/ KNODE(1100,4),XLBND(1100),TBND(1100,50),
* PSOL(1100,50)
COMMON /NLST/ XKMAX(50)
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
D l M E N S I O N  I S O L B 2 ( 5 0 ), X L B N ( 5 0 ), X B 2 1 ( 5 0 , 5 0 ) , S L B 2 1 ( 5 0 )
C
C  CALCULATE LB2
C
LB2 = 0.
DO 10 I = 1,NCUS
  LB2P = 0.
  DO 20 J = 1,NFAC
    OBJ2(J) = RAMB(I,J)
  20 CONTINUE
  CALL KNAP(CRHS,XR1MN,OBJ2,ISOLB2)
  DO 30 K = 1,NFAC
    SLB21(K) = ISOLB2(K) + XLBN(K)
  30 CONTINUE
  CALL COMPU(0.,LB2P,SLB21)
  DO 40 K = 1,NFAC
    XB21(I,K) = SLB21(K)
  40 CONTINUE
LB2 = LB2 + LB2P
10 CONTINUE
DO 50 J = 1,NFAC
  A = 0.
  DO 60 I = 1,NCUS
    60 A = A + XB21(I,J)
  SOLB2(J) = A / FLOAT(NCUS)
50 CONTINUE
C
C  SUBROUTINE CLB3(LB3,XLBN,XREMNXOPTS,VAL2)
C
Appendix A. Listing of FORTRAN Code
C THIS SUBROUTINE CALCULATES THE THIRD LOWER BOUND, BASED ON THE
C CONTINUOUS RELAXATION OF THE PROBLEM.
C
REAL LB3

COMMON /NLST/ XKMAX(50)
COMMON /ITER/ KBRN, IBVAL, IK, IK2, NF2, CRHS, IP, IBP
COMMON /KNTR/ NFATH(4), KOUNTN, FATH, NLIST, NINCSL
COMMON /PVAR/ NFAC, NCUS, COST(50), RHS
COMMON /INIT/ PROB(50, 50)
COMMON /LAMB/ RAMBJ(50), RAMB(50, 50), RATX(50)
COMMON /KPSK/ IRI(50)
COMMON /LINE/ XMINL, DJ(50)
COMMON /CLBN/ CLBN(50)
COMMON /MINC/ CJMIN
COMMON /SHIRC/ CSTAR
DIMENSION PART(50, 50), PARTL(50), RMAX(50), OPTS(50), XLBN(50)
DIMENSION SLB2(50), CB1SOL(50), XSOLB2(50), RLB1(50), RAT(30)
DIMENSION XOPTS(50), XASOL(50), XREM(50), TREMN(50)
DIMENSION IASOL(50), RATIO(50)
DIMENSION XB1SOL(50)
DOUBLE PRECISION XMUT1, XLOG, OBJJ, OLDJOB, X3
EPS = 0.0001
XMINL = 0.
OBJ3 = 0. DO
XRH = CRHS
DO 85 J = 1, NFAC
CLBN1(J) = 0.
RLB1(J) = RAMBJ(J) * COST(J)
85 CONTINUE
CALL LINK(XRH, XREM, RLB1, CLBN1)
DO 20 J = 1, NFAC
CLBN1(J) = CLBN1(J) + XLBN(J)
20 CONTINUE
CALL COMPB1(CLBN1, LB3)
IF(LB3 .GE. CSTAR .AND. NLIST .GT. 1) GO TO 77
OLDLAM = 1.

Appendix A. Listing of FORTRAN Code
DO 163 KJ = 1,NFAC
  DJ(KJ) = 0.
163 CONTINUE
  KLOOP = 0
  VINC5 = 0
  XLAM = 999.
  TOBJ = 0.
  ZZX = 0.
  DO 45 J = 1,NFAC
  DJ(J) = 0.
  OPTS(J) = CLBN1(J)
45 CONTINUE

C
C THIS IS THE MAIN LOOP FOR FINDING OPTIMAL LP SOLUTION
C
100 KLOOP = KLOOP + 1
C
C FIND THE PARTIAL DERIVATIVES
C
    DO 310 I = 1,NCUS
      DO 310 J = 1,NFAC
        PART(I,J) = 0.
310 CONTINUE
    DO 315 K = 1,NFAC
      TPART = 0.
      DO 320 I = 1,NCUS
        XMUT1 = 1.D0
        DO 325 J = 1,NFAC
          IF(OPTS(J).LT.EPS) GO TO 325
          XMUT1 = XMUT1 * DBLE(PROB(I,J))**DBLE(OPTS(J))
325 CONTINUE
      XLOG = DBLE(ALOG10(PROB(I,K)))
      PART(I,K) = XMUT1*XLOG
      TPART = TPART + PART(I,K)
320 CONTINUE
    PARTL(K) = TPART*(-1.)
RAT(K) = PARTL(K)/COST(K)

315 CONTINUE

C

C USE THE PARTIALS TO SOLVE THE LINEAR KNAPSACK

C

XRH = CRHS
CALL LINK(XRH,XREM,XASOL,RAT)
OLDOBJ = OBJ3
OBJ3 = 0.0
DO 335 J = 1,NFAC
OBJ3 = OBJ3 + (-1.0)*DBLE((XASOL(J) + XLBN(J))*PARTL(J))
* + DBLE(OPTS(J)*PARTL(J))
335 CONTINUE

XMINL = 999.
X3 = 999.0
IF(OLDOBJ.GT.EPS)X3 = ((OLDOBJ-OBJ3)/OLDOBJ)*100.
IF(DABS(OBJ3).LT.0.01)GO TO 101
IF(KLOOP.GT.4)GO TO 101
DO 40 J = 1,NFAC
DJ(J) = XASOL(J) + XLBN(J)-OPTS(J)
IF(ABS(DJ(J)).LT.EPS)GO TO 40
RMAX(J) = XKMAX(J)-OPTS(J)
IF(DJ(J).LT.0.)RMAX(J) = OPTS(J)-XLBN(J)
IF(RMAX(J).LT.(-1.* EPS))GO TO 102
XLAM = ABS(RMAX(J)/DJ(J))
IF(XLAM.LT.XMINL)XMINL = XLAM
40 CONTINUE

CALL FITQ(OPTS,OBJ,OPL,INDC,OLDLAM)
OLDLAM = OPL
DO 55 J = 1,NFAC
OPTS(J) = OPTS(J) + OPL*XMINL*DJ(J)
55 CONTINUE

GO TO 100

101 IF(KLOOP.EQ.1) CALL COMPU(ZZX,OBJ,OPTS,0)
104 LB3 = OBJ + OBJ3
IF(LB3.LT.0.)LB3 = 0.
GO TO 68

C
C FIND AN INCUMBENT BASED ON THE CONTINUOUS SOLUTION
C
77 DO 47 J = 1,NFAC
   XOPTS(J) = INT(CLBN1(J))
47 CONTINUE
GO TO 59
68 DO 10 J = 1,NFAC
   XOPTS(J) = INT(OPTS(J))
10 CONTINUE
59 XRH = RHS
   DO 1 J = 1,NFAC
      IF(XOPTS(J).LT.EPS)GO TO 1
      XRH = XRH-COST(J)*XOPTS(J)
      IF(XRH.LT.CJMIN)GO TO 4
1 CONTINUE
   DO 2 KFAC = 1,NFAC
   DO 3 J = 1,NFAC
      IF(IRJ(J).NE.KFAC)GO TO 3
      IF(COST(J).GT.XRH)GO TO 2
      MAXA = XKMAX(J)-XOPTS(J)
      MAXB = XRH/COST(J)
      IF(MAXB.LT.MAXA)MAXA = MAXB
      XOPTS(J) = XOPTS(J) + MAXA
      XRH = XRH-COST(J)*MAXA
      IF(XRH.LT.CJMIN)GO TO 4
3 CONTINUE
2 CONTINUE
4 CALL COMPU(O.,VAL2,XOPTS,O)
RETURN
102 WRITE(6,1002)
1002 FORMAT('ERROR - RMAX IS LESS THAN ZERO')
GO TO 104

Appendix A. Listing of FORTRAN Code
END

C

SUBROUTINE USER

C

THIS ROUTINE PREPARES THE INPUT FILE FOR THE MPSX LINEAR PROGRAMMING
SOFTWARE. (NOT CALLED IN THIS VERSION OF THE PROGRAM.)

C

DIMENSION AND INITIALIZE VARIABLES

C

INTEGER*2 RTYPE, RNAME, VNAME
COMMON /PVAR/ NFAC, NCUS, COST(50), RHS
DIMENSION THETA(5,80), RHSM(5,80), XVAL(80), XCOEFF(5,80), XCOST(80)
DIMENSION RTYPE(4), VNAME(2), RNAME(7), IPSOL(80)
DATA RTYPE/'N ', 'G ', 'L ', 'E'/
DATA RNAME/'OB', 'KN', 'LB', 'UB', 'AX', 'BX', 'CX'/
DATA VNAME/'X1', 'YJ'/
DATA XVAL/ 80 * 1. /
M = MCONST

C

FIND CORRESPONDING VALUES OF X VARIABLES AT THIS SOLUTION

C

DO 840 I = 1, NCUS
   DO 840 J = 1, NFAC
      XVAL(I) = XVAL(I)*(PROB(I,J)**IPSOL(J))
840 CONTINUE

C

LET FIRST THETA EQUAL X SOLUTION VALUE
OTHER THETAS ARE +/- 5% INCREMENTS

C

THETA IS AN (M X NCUS) MATRIX

C

DO 841 I = 1, NCUS
   THETA(I,1) = XVAL(I)
   DO 842 K = 2, M
      L = K/2
542 CONTINUE
\[ \Theta(K,1) = x_{val}(1) + x_{val}(1)*(L*0.05)*(-1)^K \]

842 CONTINUE
841 CONTINUE

C
C FIND RHS VECTOR FOR THE M CONSTRAINTS
C
DO 843 I = 1,NCUS
DO 843 NM = 1,M
RHSM(NM,I) = ALOG(\Theta(NM,1))-1
843 CONTINUE

C
C FIND THE COEFFICIENTS OF THE X VARIABLES
C
DO 844 I = 1,NCUS
DO 844 J = 1,M
XCOEFF(J,I) = -1./\Theta(J,I)
844 CONTINUE

C
C WRITE THE INFORMATION TO A MPSX FILE
C
WRITE(7,801)
801 FORMAT('NAME   MLBCD
C
C INPUT OF ROW DATA
C
WRITE(7,802)
802 FORMAT('ROWS
C
C THE OBJECTIVE FCN ROW AND THE KNAPSACK CONSTRAINT
C
I1 = 0
WRITE(7,803)RTYPE(1),RNAME(1),I1,I1
WRITE(7,803)RTYPE(3),RNAME(2),I1,I1
803 FORMAT( ','A2,' ','A2,I1,I1,' )
C
C THE ROWS FOR THE UPPER AND LOWER Bounds ON THE Y VARIABLES

K = 2
KK = 3
DO 845 I = 1,2
DO 846 J = 1,NFAC
WRITE(7,803)RTYPE(K),RNAME(KK),II,J
846 CONTINUE
K = 3
KK = 4
845 CONTINUE

C

C THE ROWS FOR EACH OF THE CONSTRUCTED CONSTRAINTS

DO 847 I = 1,NCUS
DO 847 J = 1,M
J1 = J + 4
WRITE(7,803)RTYPE(3),RNAME(J1),II,I
847 CONTINUE

C

C INPUT OF COLUMN DATA

WRITE(7,805)
805 FORMAT('COLUMNS

C

COLUMNS FOR THE X VARIABLES

XWT = 1.
DO 848 I = 1,NCUS
WRITE(7,807)VNAME(I),II,1,RNAME(I),II,1,XWT

C

C

COLUMNS FOR THE X VARIABLES IN THE CONSTRUCTED CONSTRAINTS

DO 851 J = 1,M
J1 = J + 4
WRITE(7,807)VNAME(I),II,1,RNAME(J1),II,1,XCOEFF(J,I)
CONTINUE

CONTINUE

C

C COLUMNS FOR ALL THE Y VARIABLES

C

THE Y VARIABLES IN THE KNAPSACK

C

DO 849 J = 1,NFAC

XCOST(J) = COST(J)

WRITE(7,807)VNAME(2),I1,J,RNAME(2),I1,J1,XCOST(J)

C

C COLUMNS FOR THE Y VARIABLES IN THE BOUNDS CONSTRAINTS

C

DO 850 I = 1,2

II = I + 2

WRITE(7,807)VNAME(2),I1,J,RNAME(II),I1,J,NWT

807 FORMAT(' ',A2,11,11,' ',A2,11,11,' ',
     • FI0.4,'•

850 CONTINUE

C

C COLUMNS FOR THE Y VARIABLES IN THE CONSTRUCTED CONSTRAINTS

C

DO 852 K = 1,NCUS

DO 852 I = 1,M

J1 = I + 4

YCOEFF = RAMB(K,J)

WRITE(7,807)VNAME(2),I1,J,RNAME(J1),I1,K,YCOEFF

852 CONTINUE

852 CONTINUE

849 CONTINUE

C

C INPUT RHS DATA

C

WRITE(7,809)

809 FORMAT('RHS

C

Appendix A. Listing of FORTRAN Code
C RHS FOR THE KNAPSACK CONSTRAINT
C
XRHS = RHS
WRITE(7,811)RNAME(2),I1,I1,XRHS
C
C RHS FOR THE CONSTRUCTED CONSTRAINTS
C
DO 853 I = 1,NCUS
DO 853 J = 1,M
J1 = J + 4
WRITE(7,811)RNAME(J1),I1,I1,RHSM(J,I)
811 FORMAT(' RSH
• A2,11,11, ',
* F6.3, ', ' )
853 CONTINUE
C
C SIGNAL END OF INPUT INFORMATION
C
WRITE(7,812)
812 FORMAT('ENDATA
RETURN
END
C
C
SUBROUTINE GENP
C
C THIS SUBROUTINE GENERATES THE PROBLEM USING A SET OF INPUT
C PARAMETERS.
C THE FOLLOWING PARAMETERS ARE USED TO DEFINE THE PROBLEM:
C NCUS IS THE NUMBER OF CUSTOMERS
C NFAC IS THE NUMBER OF FACILITIES
C RANG IS THE RANGE OF COVERING PROBABILITIES
C XMID IS THE MIDPOINT OF THE COVERING PROBABILITIES
C CRAN IS THE RANGE OF COST COEFFICIENTS
C CMID IS THE MIDPOINT OF THE COVERING PROBABILITIES
C
COMMON /PVAR/ NFAC,NCUS,COST(50),XRHS

Appendix A. Listing of FORTRAN Code
COMMON /NLST/ XKMAX(50)
COMMON /INIT/ PROB(50,50)
COMMON /PRAM/ RANG,XMID,CRAN,CMID
COMMON /MINC/ CJMIN
DOUBLE PRECISION DSEED
DSEED = 9514726.
READ(5,101)NCUS,NFAC,RANG,XMID
CRAN = 0.30
CMID = 0.50
XMC = 0.
CMIN = CMID-CRAN*0.5
CMAX = CMID+CRAN*0.5
CJMIN = CMIN*10.
J = 0
10 XNDX = GGUBFS(DSEED)
   IF(XNDX.LT.CMIN.OR.XNDX.GT.CMAX)GO TO 10
   J = J + 1
   COST(J) = FLOAT(INT(XNDX*100.))/10.
   IF(COST(J).GT.XMC)XMC = COST(J)
   IF(J.LT.NFAC)GO TO 10
   RHS = XMC*4.
   J = 0
12 XNDX = GGUBFS(DSEED)
   IF(XNDX.LT.0.4)GO TO 12
   J = J + 1
   XKMAX(J) = INT((RHS/COST(J))*XNDX)
   IF(J.LT.NFAC)GO TO 12
   XMIN = XMID-RANG*0.5
   XMAX = XMID+RANG*0.5
   DO 20 I = 1,NCUS
      J = 0
20 XNDX = GGUBFS(DSEED)
   IF(XNDX.LT.XMIN.OR.XNDX.GT.XMAX)GO TO 20
   J = J + 1
   PROB(I,J) = XNDX
   IF(J.LT.NFAC)GO TO 20

Appendix A. Listing of FORTRAN Code
SUBROUTINE LINK(XRT,XREMN,RATIO,XASOL)

C THIS SUBROUTINE FINDS A SOLUTION TO THE LINEAR CONTINUOUS
C KNAPSACK PROBLEM.

C COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /MINC/ CJMIN
DIMENSION XASOL(50),XREMN(50),OBJ(50)

C FIND RATIOS FOR OBJ AND COST

C DIMENSION XXR(50),RATIO(50)
EPS = 0.0100
XRH = XRT
DO 418 K1 = 1,NFAC
XXR(K1) = RATIO(K1)
XASOL(K1) = 0.
418 CONTINUE

KVAR = 0
420 XXMAX = XXR(1)
KNUM = 1
KVAR = KVAR + 1
DO 419 K1 = 2,NFAC
IF(XXR(K1).LE.XXMAX)GO TO 419
XXMAX = XXR(K1)
KNUM = (K1)
419 CONTINUE

C OBTAIN THE SOLUTION TO THE INTEGER KNAPSACK PROBLEM

Appendix A. Listing of FORTRAN Code
XMX = XREM(KNUM)
XMX1 = XRH / COST(KNUM)
IF(XMX.LT.XMX1) XMX1 = XMX
XASOL(KNUM) = XMX1
XRH = XRH - (XASOL(KNUM) * COST(KNUM))
IF(XRH.LT.EPS)GO TO 422
IF(KVAR.EQ.NFAC)GO TO 422
XXR(KNUM) = -1.0
GO TO 420
422 RETURN
END

C
C
C SUBROUTINE CKIN(TSOLN)
C
C THIS SUBROUTINE CHECKS TO SEE IF AN INTEGER SOLUTION HAS BEEN
C FOUND.
C
C COMMON /PVAR/ NFAC, NCUS, COST(SO), RHS
COMMON /CHKI/ KINC
DIMENSION TSOLN(SO)
EPS = 0.0010
KINC = 1
DO 10 J = 1, NFAC
TMP = AMOD(TSOLN(J), 1.)
IF(TMP.GT.EPS.AND.TMP.LT.(1.-EPS))GO TO 11
10 CONTINUE
RETURN
11 KINC = 0
RETURN
END

C
C SUBROUTINE KNAP(CKRHS, XKBND, RAT, OBJ, INTSLQ, IMAX, QSTAR)
C
C THIS SUBROUTINE SOLVES THE INTEGER KNAPSACK MAXIMIZATION PROBLEM
C USING THE CONTINUOUS SOLUTIONS TO FIND A LOWER BOUND AT EACH NODE.

C

COMMON /NLST/ XKMAX(50)
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /INIT/ PROB(50,50)
COMMON /CHKI/ KINC
COMMON /INCB/ INCSOL(50)
COMMON /LAMB/ RAMBJ(50),RAMB(50,50),RATX(50)
COMMON /KPSK/ IRJ(50)
DIMENSION QUBND(500),QLST(500,52),XLPANS(50),FRAC(50),QMSOL(50)
DIMENSION IBND(50),OBJ(50),INTSLQ(50),XKBND(50),LEVEL(500)
DIMENSION XREMN(50),RAT(50),XASOL(50),XLEFT(50)
EPS = 0.001

C

C INITIALIZE VARIABLES

C

DO 10 I = 1,500
QUBND(I) = 0.
LEVEL(I) = 0
DO 10 J = 1,NFAC
QLST(I,J) = 0.
IBND(J) = 0
XREMN(J) = XKBND(J)
10 CONTINUE

INCQ = 0
ITR = 1
CRHS = CKRHS

C

C CALL LINEAR KNAPSACK ROUTINE - OBTAIN LP SOLUTION VALUE

C

CALL LINK(CRHS,XKBND,RAT,XLPANS)

C

C FIND OBJECTIVE VALUE OF THE LP SOLUTION

C

QSTAR = 0.
QOBJ = 0.
DO 40 J = 1,NFAC
  IF(XLPANS(J).GE.EPS) QOBJ = QOBJ + XLPANS(J)*OBJ(J)
40 CONTINUE

C
C SET ROUNDED LP SOLUTION AS INITIAL INCUMBENT
C
DO 45 J = 1,NFAC
  INTSLQ(J) = INT(XLPANS(J))
  IF(INTSLQ(J).NE.0) QSTAR = QSTAR + FLOAT(INTSLQ(J))*OBJ(J)
45 CONTINUE

C SEE IF LP SOLUTION WAS INTEGER - IF SO, IT IS OPTIMAL
C
CALL CKIN(XLPANS)
IF(KINC.NE.1)GO TO 46
RETURN

C OTHERWISE, FIND THE MOST FRACTIONAL VARIABLE
C
46 FMAX = 0.
DO 70 J = 1,NFAC
  FRAC(J) = AMOD(XLPANS(J),1.)
  IF(FRAC(J).GT.0.5) FRAC(J) = 1. - FRAC(J)
  IF(FRAC(J).LT.FMAX) GO TO 70
  FMAX = FRAC(J)
  IFBR = J
70 CONTINUE

C
C STORE BRANCHING VAR, VALUE, AND MIN AND MAX BOUNDS FOR VARIABLES
C
QLST(ITR,1) = 1.
LEVEL(1) = 1
QLST(ITR,2) = IFBR*100. + INT(XLPANS(IFBR))
DO 80 J = 1,NFAC
  J2 = J + 2
  QLST(ITR,J2) = FLOAT(IBND(J))*10000. + XREMN(J)

Appendix A. Listing of FORTRAN Code  76
QUBND(ITR) = QOBJ
80 CONTINUE

C
C THIS IS THE MAIN PROGRAM LOOP - INCREMENT COUNTER
C
QBRN = 1
1 ITR = ITR + 1
91 QMAXB = 0.
   DO 95 K = 1, ITR
      IF(QLST(K,1).LE.EPS) GO TO 95
      IF(QUBND(K).LE.QMAXB) GO TO 95
      QMAXB = QUBND(K)
      QBRN = K
95 CONTINUE
   IF(QMAXB.LT.EPS) RETURN
   IF(LEVEL(QBRN).GT.IMAX) GO TO 6

C
C RETRIEVE BRANCHING VARIABLE AND VALUE
C
CRHS = CKRHS
QTEM2 = QLST(QBRN, 2)
IQVAR = (INT(QTEM2))/100
IQVAL = INT(QTEM2) - 100 * IQVAR

C
C RETRIEVE MIN AND MAX BOUNDS ON VARIABLES
C
   DO 100 J = 1, NFAC
      J2 = J + 2
      QTEM = QLST(QBRN, J2)
      IBND(J) = INT(QTEM)/10000
      XREMN(J) = INT(QTEM) - IBND(J) * 10000
      XLEFT(J) = INT(XREMN(J)) - IBND(J)
      IF(J.NE.IQVAR) GO TO 100
      XREMN(J) = FLOAT(IQVAL)
      XLEFT(J) = INT(XREMN(J)) - IBND(J)
      IF(XREMN(J).LT.0.) GO TO 4

Appendix A. Listing of FORTRAN Code
100 CONTINUE
DO 105 J = 1,NFAC
IF(IBND(J).NE.0)CRHS = CRHS-FLOAT(IBND(J))•COST(J)
105 CONTINUE
IF(CRHS.LT.0.01)GO TO 4
C
C SOLVE THE LINEAR KNAPSACK
C
CALL LINK(CRHS,XLEFT,RAT,XASOL)
NEW1 = 1
2 QOBJ = 0.
DO 110 J = 1,NFAC
XLPANS(J) = FLOAT(IBND(J)) + XASOL(J)
IF(XLPANS(J).GE.EPS)QOBJ = QOBJ + XLPANS(J)•OBJ(J)
110 CONTINUE
C
C FATHOM IF LESS THAN QSTAR
C
IF(QOBJ.GE.QSTAR)GO TO 17
QUBND(QBRN) = 0.
QLST(QBRN,1) = 0.
GO TO 4
C
C SEE IF LP SOLUTION IS INTEGER
C
17 CALL CKIN(XLPANS)
IF(KINC.EQ.0)GO TO 135
C
C IF SO,FATHOM NODE AND TRY TO UPDATE INCUMBENT
C
QUBND(QBRN) = 0.
QLST(QBRN,1) = 0.
IF(QOBJ.LE.QSTAR)GO TO 4
DO 120 J = 1,NFAC
INTSLQ(J) = INT(XLPANS(J))
120 CONTINUE
QSTAR = QOBJ

C

C IF INCUMBENT IS UPDATED, TRY TO FATHOM OTHER NODES

C

DO 130 K = 1,ITR
IF(QUBND(K).GE.QSTAR)GO TO 130
QUBND(K) = 0.
QLST(K,1) = 0.
130 CONTINUE
GO TO 4

135 FMAX = 0.
DO 140 J = 1,NFAC
FRAC(J) = AMOD(XLPANS(J),1.)
IF(FRAC(J).GT.0.50)FRAC(J) = 1.-FRAC(J)
IF(FRAC(J).LE.FMAX)GO TO 140
FMAX = FRAC(J)
IFBR = J
140 CONTINUE

C

C STORE VALUES

C

DO 145 I = 1,ITR
IF(QLST(I,1).LT.EPS)GO TO 146
145 CONTINUE

146 NSPOT = 1
LEVEL(NSPOT) = LEVEL(QBRN) + 1
QLST(NSPOT,1) = 1.
QLST(NSPOT,2) = IFBR*100. + INT(XLPANS(IFBR))
DO 150 J = 1,NFAC
J2 = J + 2
QLST(NSPOT,J2) = FLOAT(IBND(J))*10000. + XORM1(J)
QUBND(NSPOT) = QOBJ
150 CONTINUE

4 IF(NEW1.EQ.2)GO TO 180
3 ITR = ITR + 1
CRHS = CKRHS
DO 160 J = 1,NFAC
IF(I.NE.IQVAR)GO TO 160
IBND(J) = IQVAL + 1
XREMN(J) = XKBN(J)
XLEFT(J) = XKBN(J)-IBND(J)
IF(XLEFT(J).LT.0.)GO TO 180
160 CONTINUE
DO 170 J = 1,NFAC
IF(IBND(J).NE.0)CRHS = CRHS-FLOAT(IBND(J))*COST(J)
170 CONTINUE
IF(CRHS.LT.0.01)GO TO 180
CALL LINK(CRHS,XLEFT,RAT,XASOL)
NEW1 = 2
GO TO 2
C
C BOTH BRANCHES OF NODE EXPLORED - SET U.B. TO 0 AND MAKE NODE INACTIVE
C
180 QUBND(QBRN) = 0.
QLST(QBRN,1) = 0.
GO TO 1
6 XMOST = -1.0
DO 61 K = 1,ITR
IF(QLST(K,1).LE.EPS)GO TO 61
IF(QUBND(K).LE.XMOST)GO TO 61
XMOST = QUBND(K)
61 CONTINUE
QSTAR = XMOST
RETURN
END
C
C
SUBROUTINE LSRCH(OPTS,TOBJ,OPL)
C
C THIS SUBROUTINE USES A QUADRATIC FIT SEARCH METHOD TO DETERMINE THE
C BEST OBJECTIVE FUNCTION VALUE OVER THE SPECIFIED INTERVAL.
C
Appendix A. Listing of FORTRAN Code 80
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /INIT/ PROB(50,50)
DIMENSION A(50),B(50),ZLAM(50),ZMU(50),OPTS(50)
XL = 0.10
ALPHA = 0.618
A(1) = 0.
B(1) = 1.
K = 1
ZLAM(1) = 0.382
ZMU(1) = 0.618
CALL COMPU(ZLAM(K),THETA1,OPTS,1)
CALL COMPU(ZMU(K),THETA2,OPTS,1)
1 IF((B(K)-A(K)).LT.XL)GO TO 6
HERE = B(K)-A(K)
IF(T1TETAl.LT.THETAl)GO TO 1
2 KK = K + 1
A(KK) = ZLAM(K)
B(KK) = B(K)
THETA1 = THETA2
ZLAM(KK) = ZMU(K)
ZMU(KK) = A(KK) + ALPHA*(B(KK)-A(KK))
CALL COMPU(ZMU(KK),THETA2,OPTS,1)
GO TO 4
3 KK = K + 1
A(KK) = A(K)
B(KK) = ZMU(K)
THETA2 = THETA1
ZMU(KK) = ZLAM(K)
ZLAM(KK) = A(KK) + (1-ALPHA)*(B(KK)-A(KK))
CALL COMPU(ZLAM(KK),THETA1,OPTS,1)
4 K = K + 1
GO TO 1
6 IF(THETAl.LT.THETAl)GO TO 7
TOBJ = THETA2
OPL = ZMU(K)
RETURN

Appendix A. Listing of FORTRAN Code
TOBJ = THETA1
OPL = ZLAM(K)
RETURN
END

C

SUBROUTINE COMPU(ZX,TOBJ,SOLN,INDI)

C

THIS SUBROUTINE COMPUTES THE VALUE OF THE OBJECTIVE FUNCTION. WHEN
C USED IN CONJUNCTION WITH THE LINE SEARCH, THE STEP LENGTH ALONG THE
C OPTIMAL DIRECTION IS ALSO CONSIDERED.
C

COMMON /PVAR/ NFAC, NCUS, COST(50), RHS
COMMON /INIT/ PROB(50, 50)
COMMON /LINE/ XMINL, DJ(50)
DOUBLE PRECISION XMUT, TOL
DIMENSION SOLN(50), TSOL(50)
TOL = 0.00000001D0
EPS = 0.001
IF(INDI.EQ.0)GO TO 31
DO 10 J = 1, NFAC
TSOL(J) = SOLN(J) + ZX*XMINL*DJ(J)
10 CONTINUE
GO TO 12
31 DO 32 J = 1, NFAC
TSOL(J) = SOLN(J)
32 CONTINUE
12 TOBJ = 0.
DO 20 I = 1, NCUS
XMUT = I.D0
DO 25 J = 1, NFAC
XMUT = XMUT*DBLE(PROB(I,J))**DBLE(TSOL(J))
IF(XMUT.LT.TOL)XMUT = 0.
25 CONTINUE
TOBJ = TOBJ + XMUT
20 CONTINUE
SUBROUTINE COMPB1(BND1, VAL2)

C THIS SUBROUTINE FINDS THE OBJECTIVE VALUE OF AN INTEGER OR CONTINUOUS
C SOLUTION TO LBI BOUNDING STRATEGY.

COMMON /PVAR/ NFAC, NCUS, COST(50), RHS
COMMON /LAMB/ RAMBJ(50), RAMB(50, 50), RATX(50)
COMMON /INIT/ PROB(50, 50)
DIMENSION BND1(50)
DOUBLE PRECISION X2

X2 = 0.
DO 20 J = 1, NFAC
  X2 = X2 + BND1(J)*RAMBJ(J)
20 CONTINUE

XPON = X2*(-1.)*NCUS

VAL2 = EXP(XPON)*NCUS
RETURN
END

SUBROUTINE FITQ(OPTS, TOBJ, OPL, INDC, OLDLAM)

C THIS SUBROUTINE CONTAINS THE GRADIENT FIT LINE SEARCH PROCEDURE.

COMMON /PVAR/ NFAC, NCUS, COST(50), RHS
COMMON /INIT/ PROB(50, 50)
DIMENSION OPTS(50)

INDC = 0
KITR = 1
A = 0.
B = OLDLAM
RI = B - A
CALL COMPU(A,THETA1,OPTS,1)
CALL COMPU(B,THETA2,OPTS,1)
C = B-0.50*RI
1 CALL COMPU(C,THETA3,OPTS,1)
IF(THETA1.GT.THETA3.AND.THETA3.LT.THETA2)GO TO 4
TOL = (THETA1 + THETA2 + THETA3)/3.*0.01
IF(ABS(THETA1-THETA2),LT.TOL.AND.ABS(THETA2-THETA3),LT.TOL,GO TO 4
IF(THETA1.GT.THETA3.AND.THETA3.GT.THETA2)GO TO 20
IF(THETA1.LT.THETA3.AND.THETA3.LT.THETA2)GO TO 25
GO TO 99
20 RI = RI/2.
A = C
THETA1 = THETA3
C = C + 0.50*RI
KITR = KITR + 1
IF(KITR.NE.4)GO TO 2
A = B
THETA1 = THETA2
B = 1.
CALL COMPU(B,THETA2,OPTS,1)
RI = B-A
C = B-(RI*.50)
2 GO TO 1
25 RI = RI/2.
B = C
THETA2 = THETA3
C = C-0.50*RI
KITR = KITR + 1
GO TO 1
4 OPL = 0.50*((C**2-B**2)*THETA1 + (B**2-A**2)*THETA3 +
* (A**2-C**2)*THETA2))/((C-B)*THETA1 + (B-A)*THETA3 + (A-C)*THETA2)
XMIN = THETA1
IF(THETA2.LT.THETA1)XMIN = THETA2
X1 = (XMIN-THETA2)/XMIN
IF(X1.LT.0.001)INDC = 1
CALL COMPU(OPL,TOBJ,OPTS,1)
RETURN
99 WRITE(6,23)
23 FORMAT(' ERROR - FUNCTION NOT CONVEX ')
RETURN
END

C

SUBROUTINE BRAN(IND,XLBN,KERR,XREMN,KINTGR,XBISOL)
C
C THIS SUBROUTINE SELECTS THE BRANCHING VARIABLE FOR THE NODE AND
C STORES THE SOLUTION VALUES FOR THE CURRENT NODE.
C
COMMON /CLBN/ CLBN1(50)
COMMON /DLST/ KNODE(1100,4),XLBND(1100),TBND(1100,50),
* PSOL(1100,50)
COMMON /NLST/ XKMAX(50)
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /BNDS/ XLB,LB1,LB3,KNBND
COMMON /KNTR/ NFAT1(4),KOUTN,FATH,NLIST,NINCSL
COMMON /ITER/ KBRN,IBVAL,IK,IK2,NF2,CRHS,IP,IBP
COMMON /BRCH/ IND2
COMMON /ZRON/ KNZRO,NZRO(50)
COMMON /LAMB/ RAMBJ(50),RAMB(50,50),RATX(50)
COMMON /KPSK/ IRJ(50)
DIMENSION FRAC(50),XLBN(50),XREMN(50),XBISOL(50)
EPS = 0.001
KERR = 0
KINTGR = 0

C
C FIND THE MOST FRACTIONAL VAR.
C
KBRN = 1
FMAX = 0.
DO 70 J = 1,NFAC
FRAC(J) = (CLBN1(J)-INT(CLBN1(J)))/EPS
IF(FRAC(J).GT.0.5)FRAC(J) = 1.-FRAC(J)

Appendix A. Listing of FORTRAN Code
IF((FRAC(J) - FMAX).LT.(-1.*EPS)) GO TO 70
IF((FRAC(J) - FMAX).GT.EPS) GO TO 72
IF(IRJ(J).GT.IRJ(KBRN)) GO TO 70
72 FMAX = FRAC(J)
    KBRN = J
70 CONTINUE

C
C USE PERMUTATION INDEX IF NO FRACTIONAL VARIABLE IS FOUND
C
    IRJMIN = 999
    MININD = 0
    IF(FMAX.GT.EPS) GO TO 552
    DO 71 J = 1, KNZRO
        IF(IRJ(NZRO(J)).EQ.IK) GO TO 71
        IF((XKMAX(NZRO(J))-1.0T(CLBN1(NZRO(J)))).LT.EPS) GO TO 71
        IF(IRJ(NZRO(J)).GT.IRJMIN) GO TO 71
        IRJMIN = IRJ(NZRO(J))
        MININD = NZRO(J)
    71 CONTINUE
    IF(MININD.NE.0) GO TO 77
    NFATH(4) = NFATH(4) + 1
    KINTGR = 1
    RETURN
77 KBRN = MININD
552 IF(IND.EQ.1) RETURN

C
C STORE ABOVE RESULTS IN LIST AND XBLND
C
    NLIST = NLIST + 1
    DO 110 KK = 1, 1100
        IF(KNODE(KK,1).EQ.0) GO TO 120
    110 CONTINUE
    WRITE(6,301)
    301 FORMAT(2X,'ERROR - NO STORAGE SPACE IS AVAILABLE')
        KERR = 1
        RETURN
120 NSPOT = KK
   KNODE(NSPOT,1) = 1
   KNODE(NSPOT,2) = KNODE(IBP,2) + 1
   KNODE(NSPOT,3) = NLIST
   KNODE(NSPOT,4) = FLOAT(KBRN * 100) + INT(CLBN1(KBRN))
   XLBND(NSPOT) = XLB
   DO 80 J = 1,NFAC
   TDND(NSPOT,J) = XLBN(J)*100 + XKMAX(J)
   PSOL(NSPOT,J) = XBISOL(J)
80 CONTINUE
RETURN
END

SUBROUTINE INCUM(IND,XLIMIT,XBISOL)
C THIS SUBROUTINE DETERMINES WHETHER A NEW INCUMBENT SOLUTION HAS BEEN FOUND.
C
REAL LB1
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /BNDS/ XLB,LB1,LB3,KNBND
COMMON /XLBD/ ZSTAR,KKLB,ORGINC
DIMENSION XBISOL(50)
EPS = 0.0001

C IF THIS IS NODE 0, LET THE LBISOL AUTOMATICALLY BE THE INCUMBENT
C
IF(IND.NE.1)GO TO 20
CALL COMPU(O.,OBJ1,XBISOL,0)
ORGINC = OBJ1
CALL UPDATE(OBJ1,1,XBISOL,XLIMIT)
GO TO 30
C
C SEE IF NEW INCUMBENT SOLUTION IS FOUND
C
Appendix A. Listing of FORTRAN Code
20 IF(LB1.GT.XLIMIT)GO TO 30
   CALL COMPU(0.,OBJ1,XBISOL,0)
   IF(OBJ1.GT.XLIMIT)GO TO 30
   CALL UPDATE(OBJ1,1,XBISOL,XLIMIT)
30 RETURN
   END

C
C
SUBROUTINE UPDATE(BND,NBND,BSOLN,XLIMIT)
C
C THIS SUBROUTINE UPDATES THE INCUMBENT SOLUTION.
C
COMMON /DLST/ KNODE(1100,4),XLBND(1100),TBND(1100,50),
* PSOL(1100,50)
COMMON /PVAR/ NFAC,NCUS,COST(50),RHS
COMMON /KNTR/ NFATH(4),KOUNTN,FATH,NLIST,NINCSL
COMMON /XLBD/ ZSTAR,KKLB,ORGINC
COMMON /INCB/ INCSOL(50)
DIMENSION BSOLN(50)
EPS = 0.0001
C
C FIND IMPROVEMENT OVER LAST INCUMBENT SOLUTION
C
   XPTG = ((ZSTAR-BND)/ZSTAR)*100.
C
C STORE VALUES FOR NEW INCUMBENT
C
   ZSTAR = BND
   XLIMIT = ZSTAR-FATH*ZSTAR
   NINCSL = KOUNTN
   DO 3 J = 1,NFAC
      INCSOL(J) = INT(BSOLN(J))
   3 CONTINUE
C
C TRY TO FATHOM NODES IN LIST USING NEW INCUMBENT
C
Appendix A. Listing of FORTRAN Code
DO 4 I = 1,NLIST
  IF(KNODE(I,1).LT.EPS)GO TO 4
  IF(XLBN(D(I)).LT.XLIMIT)GO TO 4
  KNODE(I,1) = 0.
  NFATH(I) = NFATH(I) + 1
4 CONTINUE
RETURN
END
Appendix B. Detailed Computational Results

The tables in this appendix give the detailed output results from each of the computational comparisons. Included in the tables is the following information:

- the problem parameters - the range of the covering probabilities, the size of the problem (number of customers by number of available facilities), and the mean of the covering probabilities
- the total number of nodes enumerated by the algorithm
- the number of nodes that were stored as active nodes
- the objective value of the optimal solution to the problem
- the node at which the optimal solution was found
- the gap between the initial incumbent solution and the best bound at node zero
in routines other than Continuous LB1, the average gap between the incumbent solution and the bound value in cases where the Continuous LB1 value did not fathom the node, but a stronger bound (either Integer LB1 or LB3 did fathom)

the percent improvement of the optimal solution over the initial incumbent solution

the percent of nodes that were fathomed

the percentage of overall fathoming that was accounted for by Continuous LB1, Integer LB1, or LB3

the CPU time, measured in seconds
### RESULTS OF INTEGER LB1 ALGORITHM - PROBLEM SET 1

<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>AVG %</th>
<th>CLB1</th>
<th>CLB3</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>15X10</td>
<td>0.50</td>
<td>527</td>
<td>264</td>
<td>173</td>
<td>12.97</td>
<td>173</td>
<td>12.97</td>
<td>0.58</td>
<td>0.23</td>
<td>2.42</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>15X10</td>
<td>0.50</td>
<td>301</td>
<td>150</td>
<td>25</td>
<td>8.55</td>
<td>25</td>
<td>8.55</td>
<td>0.52</td>
<td>0.23</td>
<td>1.56</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>20X12</td>
<td>0.75</td>
<td>2123</td>
<td>1063</td>
<td>351</td>
<td>6.16</td>
<td>351</td>
<td>6.16</td>
<td>0.42</td>
<td>0.14</td>
<td>9.50</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>15X10</td>
<td>0.50</td>
<td>1179</td>
<td>589</td>
<td>35</td>
<td>8.70</td>
<td>35</td>
<td>8.70</td>
<td>0.29</td>
<td>0.27</td>
<td>5.94</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>20X12</td>
<td>0.25</td>
<td>4447</td>
<td>2223</td>
<td>61</td>
<td>42.01</td>
<td>61</td>
<td>42.01</td>
<td>0.17</td>
<td>0.27</td>
<td>25.39</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>15X10</td>
<td>0.65</td>
<td>1869</td>
<td>934</td>
<td>19</td>
<td>16.98</td>
<td>19</td>
<td>16.98</td>
<td>0.15</td>
<td>0.27</td>
<td>8.98</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>10X10</td>
<td>0.35</td>
<td>3091</td>
<td>1571</td>
<td>605</td>
<td>31.84</td>
<td>605</td>
<td>31.84</td>
<td>0.09</td>
<td>0.19</td>
<td>11.80</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>20X12</td>
<td>0.35</td>
<td>12621</td>
<td>6310</td>
<td>151</td>
<td>36.81</td>
<td>151</td>
<td>36.81</td>
<td>0.04</td>
<td>0.16</td>
<td>60.48</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>20X12</td>
<td>0.50</td>
<td>10737</td>
<td>5371</td>
<td>795</td>
<td>27.99</td>
<td>795</td>
<td>27.99</td>
<td>0.03</td>
<td>0.20</td>
<td>57.02</td>
</tr>
</tbody>
</table>
### RESULTS OF CONTINUOUS LBI ALGORITHM - PROBLEM SET I

<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>15X10</td>
<td>0.50</td>
<td>1499</td>
<td>749</td>
<td>2260E+00</td>
<td>185</td>
<td>34.68</td>
<td>3.85</td>
<td>0.26</td>
<td>2.220</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>15X10</td>
<td>0.50</td>
<td>721</td>
<td>361</td>
<td>2238E+00</td>
<td>27</td>
<td>19.98</td>
<td>10.42</td>
<td>0.31</td>
<td>1.030</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>20X12</td>
<td>0.75</td>
<td>3455</td>
<td>1728</td>
<td>2957E+01</td>
<td>205</td>
<td>14.17</td>
<td>10.87</td>
<td>0.23</td>
<td>7.730</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>15X10</td>
<td>0.50</td>
<td>1827</td>
<td>913</td>
<td>2328E+00</td>
<td>61</td>
<td>46.83</td>
<td>28.14</td>
<td>0.21</td>
<td>3.070</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>20X12</td>
<td>0.25</td>
<td>8259</td>
<td>4129</td>
<td>2038E-02</td>
<td>107</td>
<td>85.25</td>
<td>52.78</td>
<td>0.19</td>
<td>20.480</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>15X10</td>
<td>0.65</td>
<td>3663</td>
<td>1831</td>
<td>1028E+01</td>
<td>19</td>
<td>19.36</td>
<td>3.75</td>
<td>0.19</td>
<td>6.720</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>10X10</td>
<td>0.35</td>
<td>5021</td>
<td>2510</td>
<td>7741E-02</td>
<td>551</td>
<td>87.14</td>
<td>65.72</td>
<td>0.11</td>
<td>7.430</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>20X12</td>
<td>0.35</td>
<td>15137</td>
<td>7568</td>
<td>3203E+01</td>
<td>163</td>
<td>70.68</td>
<td>61.21</td>
<td>0.05</td>
<td>48.510</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>20X12</td>
<td>0.50</td>
<td>14263</td>
<td>7131</td>
<td>2718E+00</td>
<td>793</td>
<td>66.12</td>
<td>48.96</td>
<td>0.05</td>
<td>46.500</td>
</tr>
</tbody>
</table>
### RESULTS OF LB3 ALGORITHM - PROBLEM SET I

<table>
<thead>
<tr>
<th>NO.</th>
<th>PROB</th>
<th>PROB RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>15X10</td>
<td>0.50</td>
<td>959</td>
<td>480</td>
<td>.2260E+00</td>
<td>157</td>
<td>1.52</td>
<td>3.85</td>
<td>0.46</td>
<td>0.13</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>15X10</td>
<td>0.50</td>
<td>211</td>
<td>106</td>
<td>.2240E+00</td>
<td>10</td>
<td>9.29</td>
<td>10.35</td>
<td>0.42</td>
<td>0.29</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>20X12</td>
<td>0.75</td>
<td>211</td>
<td>105</td>
<td>.2957E+01</td>
<td>18</td>
<td>9.80</td>
<td>7.22</td>
<td>0.21</td>
<td>0.64</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>15X10</td>
<td>0.50</td>
<td>347</td>
<td>180</td>
<td>.2328E+00</td>
<td>101</td>
<td>16.52</td>
<td>28.14</td>
<td>0.13</td>
<td>0.60</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>20X12</td>
<td>0.25</td>
<td>1205</td>
<td>622</td>
<td>.2038E-02</td>
<td>547</td>
<td>59.94</td>
<td>52.78</td>
<td>0.11</td>
<td>0.59</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>15X10</td>
<td>0.65</td>
<td>327</td>
<td>172</td>
<td>.1028E+01</td>
<td>156</td>
<td>29.68</td>
<td>3.75</td>
<td>0.07</td>
<td>0.56</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>10X10</td>
<td>0.35</td>
<td>137</td>
<td>72</td>
<td>.7741E-02</td>
<td>21</td>
<td>64.48</td>
<td>65.72</td>
<td>0.06</td>
<td>0.78</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>20X12</td>
<td>0.35</td>
<td>77</td>
<td>40</td>
<td>.3203E-01</td>
<td>14</td>
<td>93.16</td>
<td>61.21</td>
<td>0.05</td>
<td>0.74</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>20X12</td>
<td>0.50</td>
<td>667</td>
<td>338</td>
<td>.2718E+00</td>
<td>374</td>
<td>63.04</td>
<td>57.75</td>
<td>0.01</td>
<td>0.51</td>
</tr>
<tr>
<td>NO.</td>
<td>RANGE</td>
<td>SIZE</td>
<td>NODES</td>
<td>NODES</td>
<td>OPT</td>
<td>OPT</td>
<td>INIT</td>
<td>%IMP</td>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>1541</td>
<td>773</td>
<td>.7522E+00</td>
<td>397</td>
<td>38.62</td>
<td>3.99</td>
<td>0.26</td>
<td>5.950</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>1585</td>
<td>792</td>
<td>.5014E+00</td>
<td>63</td>
<td>39.03</td>
<td>23.79</td>
<td>0.23</td>
<td>4.220</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>3187</td>
<td>1593</td>
<td>.5856E+01</td>
<td>1</td>
<td>3.89</td>
<td>0.0</td>
<td>0.25</td>
<td>12.910</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>2381</td>
<td>1190</td>
<td>.8565E+00</td>
<td>199</td>
<td>31.67</td>
<td>14.96</td>
<td>0.19</td>
<td>10.530</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>12319</td>
<td>6159</td>
<td>.6546E-02</td>
<td>813</td>
<td>98.11</td>
<td>63.62</td>
<td>0.12</td>
<td>44.600</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>2553</td>
<td>1276</td>
<td>.2387E+01</td>
<td>165</td>
<td>29.13</td>
<td>23.53</td>
<td>0.22</td>
<td>9.630</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>7745</td>
<td>3822</td>
<td>.1259E+00</td>
<td>179</td>
<td>63.83</td>
<td>47.48</td>
<td>0.01</td>
<td>38.930</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>16707</td>
<td>8353</td>
<td>.4790E-01</td>
<td>653</td>
<td>79.22</td>
<td>70.15</td>
<td>0.04</td>
<td>63.890</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>16409</td>
<td>9010</td>
<td>.6228E+00</td>
<td>179</td>
<td>55.82</td>
<td>33.12</td>
<td>0.02</td>
<td>76.260</td>
</tr>
</tbody>
</table>

RESULTS OF CONTINUOUS LB1 ALGORITHM - PROBLEM SET II
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>FATH</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>661</td>
<td>357</td>
<td>0.7522E+00</td>
<td>209</td>
<td>9.09</td>
<td>0.55</td>
<td>0.11</td>
<td>4.470</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>869</td>
<td>444</td>
<td>0.5014E+00</td>
<td>81</td>
<td>10.51</td>
<td>0.44</td>
<td>0.16</td>
<td>4.660</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>2321</td>
<td>1160</td>
<td>0.5856E+01</td>
<td>1</td>
<td>5.46</td>
<td>0.46</td>
<td>0.12</td>
<td>14.200</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>1515</td>
<td>757</td>
<td>0.8565E+00</td>
<td>357</td>
<td>13.12</td>
<td>0.10</td>
<td>0.35</td>
<td>11.630</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>6435</td>
<td>3232</td>
<td>0.6546E-02</td>
<td>741</td>
<td>40.99</td>
<td>0.09</td>
<td>0.25</td>
<td>40.830</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>1539</td>
<td>778</td>
<td>0.2387E+01</td>
<td>195</td>
<td>16.82</td>
<td>0.15</td>
<td>0.30</td>
<td>10.300</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>7375</td>
<td>3687</td>
<td>0.1259E+00</td>
<td>167</td>
<td>29.49</td>
<td>0.01</td>
<td>0.14</td>
<td>47.390</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>14013</td>
<td>7006</td>
<td>0.4790E-01</td>
<td>819</td>
<td>34.15</td>
<td>0.04</td>
<td>0.15</td>
<td>77.920</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>13041</td>
<td>6520</td>
<td>0.6228E+00</td>
<td>143</td>
<td>25.94</td>
<td>0.01</td>
<td>0.18</td>
<td>87.220</td>
</tr>
</tbody>
</table>

RESULTS OF INTEGER LB1 ALGORITHM - PROBLEM SET II
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>1019</td>
<td>510</td>
<td>.7522E+00</td>
<td>765</td>
<td>1.11</td>
<td>3.99</td>
<td>0.35</td>
<td>0.14</td>
<td>39.99</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>445</td>
<td>224</td>
<td>.5014E+00</td>
<td>49</td>
<td>11.65</td>
<td>23.79</td>
<td>0.22</td>
<td>0.47</td>
<td>14.15</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>123</td>
<td>61</td>
<td>.5856E+01</td>
<td>1</td>
<td>11.00</td>
<td>0.0</td>
<td>0.11</td>
<td>0.0</td>
<td>17.75</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>349</td>
<td>177</td>
<td>.8565E+00</td>
<td>52</td>
<td>29.04</td>
<td>14.96</td>
<td>0.02</td>
<td>0.75</td>
<td>32.36</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>791</td>
<td>400</td>
<td>.6546E-02</td>
<td>276</td>
<td>67.28</td>
<td>63.62</td>
<td>0.04</td>
<td>0.68</td>
<td>51.49</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>159</td>
<td>85</td>
<td>.2387E+01</td>
<td>90</td>
<td>37.41</td>
<td>23.53</td>
<td>0.10</td>
<td>0.66</td>
<td>18.57</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>219</td>
<td>110</td>
<td>.1259E+00</td>
<td>10</td>
<td>85.47</td>
<td>60.47</td>
<td>0.01</td>
<td>0.76</td>
<td>48.05</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>133</td>
<td>70</td>
<td>.4790E-01</td>
<td>20</td>
<td>85.44</td>
<td>70.15</td>
<td>0.06</td>
<td>0.78</td>
<td>20.70</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>345</td>
<td>173</td>
<td>.6228E+00</td>
<td>18</td>
<td>58.31</td>
<td>54.90</td>
<td>0.01</td>
<td>0.51</td>
<td>82.78</td>
</tr>
</tbody>
</table>

RESULTS OF LB3 ALGORITHM - PROBLEM SET II
## RESULTS OF CONTINUOUS LB1 ALGORITHM - PROBLEM SET III

<table>
<thead>
<tr>
<th>NO</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>8x20</td>
<td>0.50</td>
<td>20117</td>
<td>10073</td>
<td>.1128E+00</td>
<td>2729</td>
<td>43.45</td>
<td>8.88</td>
<td>0.24</td>
<td>45.790</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>7x25</td>
<td>0.50</td>
<td>1567</td>
<td>783</td>
<td>.3497E-01</td>
<td>3</td>
<td>15.95</td>
<td>8.10</td>
<td>0.43</td>
<td>2.900</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>10x15</td>
<td>0.75</td>
<td>6181</td>
<td>3090</td>
<td>.1374E+01</td>
<td>243</td>
<td>16.65</td>
<td>12.81</td>
<td>0.26</td>
<td>12.700</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>8x20</td>
<td>0.50</td>
<td>10009</td>
<td>5004</td>
<td>.6720E-01</td>
<td>89</td>
<td>53.95</td>
<td>24.43</td>
<td>0.36</td>
<td>22.230</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>7x25</td>
<td>0.25</td>
<td>ERROR</td>
<td>--</td>
<td>NO STORAGE SPACE IS AVAILABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>10x15</td>
<td>0.65</td>
<td>3193</td>
<td>1596</td>
<td>.2957E+00</td>
<td>121</td>
<td>49.76</td>
<td>35.74</td>
<td>0.21</td>
<td>5.750</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>5x10</td>
<td>0.35</td>
<td>3745</td>
<td>1872</td>
<td>.3205E-02</td>
<td>91</td>
<td>98.50</td>
<td>87.07</td>
<td>0.16</td>
<td>4.780</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>7x25</td>
<td>0.35</td>
<td>ERROR</td>
<td>--</td>
<td>NO STORAGE SPACE IS AVAILABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>5x10</td>
<td>0.50</td>
<td>2013</td>
<td>1006</td>
<td>.1940E-01</td>
<td>77</td>
<td>65.95</td>
<td>51.81</td>
<td>0.24</td>
<td>2.150</td>
</tr>
</tbody>
</table>

**Notes:**
- ERROR indicates that the algorithm encountered an error due to insufficient storage space.
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN NODE</th>
<th>GAP</th>
<th>FATH</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>8X20</td>
<td>0.50</td>
<td>16709</td>
<td>8379</td>
<td>.1128E+00</td>
<td>3047</td>
<td>5.72</td>
<td>0.45</td>
<td>0.07</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>7X25</td>
<td>0.50</td>
<td>1367</td>
<td>683</td>
<td>.3497E-01</td>
<td>3</td>
<td>9.05</td>
<td>0.85</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>10X15</td>
<td>0.75</td>
<td>3931</td>
<td>1974</td>
<td>.1374E+01</td>
<td>929</td>
<td>5.94</td>
<td>0.36</td>
<td>0.15</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>8X20</td>
<td>0.50</td>
<td>8143</td>
<td>4071</td>
<td>.6720E-01</td>
<td>37</td>
<td>8.06</td>
<td>0.68</td>
<td>0.08</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>7X25</td>
<td>0.25</td>
<td>ERROR</td>
<td>NO STORAGE SPACE IS AVAILABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>10X15</td>
<td>0.65</td>
<td>2163</td>
<td>1081</td>
<td>.2957E+00</td>
<td>101</td>
<td>10.55</td>
<td>0.29</td>
<td>0.13</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>5X10</td>
<td>0.35</td>
<td>2571</td>
<td>1285</td>
<td>.3205E-02</td>
<td>153</td>
<td>40.78</td>
<td>0.13</td>
<td>0.24</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>7X25</td>
<td>0.35</td>
<td>ERROR</td>
<td>NO STORAGE SPACE IS AVAILABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>5X10</td>
<td>0.50</td>
<td>1215</td>
<td>611</td>
<td>.1940E-01</td>
<td>43</td>
<td>22.78</td>
<td>0.31</td>
<td>0.22</td>
</tr>
</tbody>
</table>

RESULTS OF INTEGER LB1 ALGORITHM - PROBLEM SET III
<table>
<thead>
<tr>
<th>NO.</th>
<th>PROB</th>
<th>NODES</th>
<th>OPT</th>
<th>AVG</th>
<th>%IMP</th>
<th>CLB1</th>
<th>CLB3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>8X20</td>
<td>0.50</td>
<td>14163</td>
<td>7101</td>
<td>3907</td>
<td>1.37</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>7X25</td>
<td>0.50</td>
<td>91</td>
<td>45</td>
<td>17</td>
<td>18.66</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>10X15</td>
<td>0.50</td>
<td>115</td>
<td>61</td>
<td>31</td>
<td>12.26</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>8X20</td>
<td>0.50</td>
<td>91</td>
<td>459</td>
<td>197</td>
<td>17.43</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>7X25</td>
<td>0.50</td>
<td>911</td>
<td>459</td>
<td>197</td>
<td>17.43</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>10X15</td>
<td>0.50</td>
<td>911</td>
<td>459</td>
<td>197</td>
<td>17.43</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>5X10</td>
<td>0.35</td>
<td>407</td>
<td>210</td>
<td>61</td>
<td>64.93</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>7X25</td>
<td>0.35</td>
<td>923</td>
<td>521</td>
<td>698</td>
<td>84.73</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>5X10</td>
<td>0.50</td>
<td>139</td>
<td>70</td>
<td>10</td>
<td>35.38</td>
</tr>
</tbody>
</table>

RESULTS OF LB3 ALGORITHM - PROBLEM SET III
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>2915</td>
<td>789</td>
<td>.7522E+00</td>
<td>1715</td>
<td>38.62</td>
<td>3.99</td>
<td>0.13</td>
<td>6.200</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>2917</td>
<td>808</td>
<td>.5014E+00</td>
<td>1271</td>
<td>39.03</td>
<td>23.79</td>
<td>0.12</td>
<td>4.230</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>5797</td>
<td>1593</td>
<td>.5856E+01</td>
<td>1</td>
<td>3.89</td>
<td>0.0</td>
<td>0.14</td>
<td>12.360</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>4321</td>
<td>1237</td>
<td>.8565E+00</td>
<td>1939</td>
<td>31.67</td>
<td>14.96</td>
<td>0.10</td>
<td>10.380</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>22307</td>
<td>6165</td>
<td>.6546E-02</td>
<td>3119</td>
<td>98.11</td>
<td>63.62</td>
<td>0.06</td>
<td>39.130</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>4673</td>
<td>1346</td>
<td>.2387E+01</td>
<td>2563</td>
<td>29.13</td>
<td>23.53</td>
<td>0.12</td>
<td>9.530</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>13847</td>
<td>3872</td>
<td>.1259E+00</td>
<td>1503</td>
<td>68.55</td>
<td>60.47</td>
<td>0.01</td>
<td>36.740</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>30241</td>
<td>8353</td>
<td>.4790E-01</td>
<td>8071</td>
<td>79.22</td>
<td>70.15</td>
<td>0.02</td>
<td>56.470</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>29937</td>
<td>8204</td>
<td>.6228E+00</td>
<td>4291</td>
<td>66.89</td>
<td>54.90</td>
<td>0.02</td>
<td>68.850</td>
</tr>
</tbody>
</table>

RESULTS OF CONTINUOUS LB1 ALGORITHM - PROBLEM SET II, DEPTH-FIRST
<table>
<thead>
<tr>
<th>No.</th>
<th>Range</th>
<th>Size</th>
<th>Mean</th>
<th>Nodes</th>
<th>Opt</th>
<th>Opt</th>
<th>Avg %</th>
<th>CLB1</th>
<th>LB1</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>777</td>
<td>388</td>
<td>.7522E+00</td>
<td>271</td>
<td>12.32</td>
<td>0.52</td>
<td>0.14</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>1125</td>
<td>562</td>
<td>.5014E+00</td>
<td>765</td>
<td>10.40</td>
<td>0.41</td>
<td>0.18</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>2321</td>
<td>1160</td>
<td>.5856E+01</td>
<td>1</td>
<td>5.46</td>
<td>0.46</td>
<td>0.12</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>1515</td>
<td>757</td>
<td>.8565E+00</td>
<td>455</td>
<td>13.97</td>
<td>0.08</td>
<td>0.37</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>14101</td>
<td>7051</td>
<td>.6546E-02</td>
<td>9429</td>
<td>40.62</td>
<td>0.06</td>
<td>0.19</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>1577</td>
<td>788</td>
<td>.2387E+01</td>
<td>243</td>
<td>18.17</td>
<td>0.14</td>
<td>0.31</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>7395</td>
<td>3697</td>
<td>.1259E+00</td>
<td>3039</td>
<td>30.48</td>
<td>0.01</td>
<td>0.14</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>19929</td>
<td>9964</td>
<td>.4790E-01</td>
<td>****</td>
<td>42.87</td>
<td>0.01</td>
<td>0.14</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>21505</td>
<td>10752</td>
<td>.6228E+00</td>
<td>****</td>
<td>29.32</td>
<td>0.02</td>
<td>0.17</td>
</tr>
</tbody>
</table>

RESULTS OF INTEGER LB1 ALGORITHM - PROBLEM SET II, DEPTH-FIRST
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>1907</td>
<td>523</td>
<td>.7522E+00</td>
<td>1875</td>
<td>1.51</td>
<td>3.99</td>
<td>0.11</td>
<td>0.70</td>
<td>42.980</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>1267</td>
<td>350</td>
<td>.5014E+00</td>
<td>909</td>
<td>11.21</td>
<td>23.79</td>
<td>0.09</td>
<td>0.78</td>
<td>21.690</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>233</td>
<td>61</td>
<td>.5856E+01</td>
<td>1</td>
<td>11.00</td>
<td>0.0</td>
<td>0.04</td>
<td>0.85</td>
<td>17.670</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>1025</td>
<td>279</td>
<td>.8565E+00</td>
<td>786</td>
<td>29.85</td>
<td>14.96</td>
<td>0.0</td>
<td>0.85</td>
<td>42.630</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>1461</td>
<td>405</td>
<td>.6546E-02</td>
<td>604</td>
<td>67.78</td>
<td>63.62</td>
<td>0.01</td>
<td>0.87</td>
<td>50.920</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>673</td>
<td>182</td>
<td>.2387E+01</td>
<td>640</td>
<td>33.31</td>
<td>23.53</td>
<td>0.01</td>
<td>0.88</td>
<td>31.910</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>537</td>
<td>145</td>
<td>.1259E+00</td>
<td>190</td>
<td>86.10</td>
<td>60.47</td>
<td>0.0</td>
<td>0.86</td>
<td>51.330</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>425</td>
<td>116</td>
<td>.4790E-01</td>
<td>288</td>
<td>86.80</td>
<td>70.15</td>
<td>0.0</td>
<td>0.90</td>
<td>30.420</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>865</td>
<td>234</td>
<td>.6228E+00</td>
<td>240</td>
<td>62.52</td>
<td>54.90</td>
<td>0.0</td>
<td>0.81</td>
<td>101.910</td>
</tr>
</tbody>
</table>

RESULTS OF LB3 ALGORITHM - PROBLEM SET II, DEPTH-FIRST
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>FATH</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>1483</td>
<td>741</td>
<td>.7835E+00</td>
<td>1</td>
<td>38.62</td>
<td>0.0</td>
<td>0.28</td>
<td>5.280</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>1301</td>
<td>650</td>
<td>.5014E+00</td>
<td>63</td>
<td>39.03</td>
<td>23.79</td>
<td>0.27</td>
<td>3.200</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>1</td>
<td>1</td>
<td>.5856E+01</td>
<td>1</td>
<td>39.03</td>
<td>0.0</td>
<td>0.0</td>
<td>0.680</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>2311</td>
<td>1155</td>
<td>.8929E+00</td>
<td>185</td>
<td>31.67</td>
<td>11.34</td>
<td>0.19</td>
<td>9.970</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>12191</td>
<td>6095</td>
<td>.6817E-02</td>
<td>181</td>
<td>98.11</td>
<td>62.12</td>
<td>0.12</td>
<td>43.160</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>2265</td>
<td>1132</td>
<td>.2387E+01</td>
<td>165</td>
<td>29.13</td>
<td>23.53</td>
<td>0.21</td>
<td>8.310</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>7709</td>
<td>3854</td>
<td>.1259E+00</td>
<td>165</td>
<td>68.55</td>
<td>60.47</td>
<td>0.01</td>
<td>38.760</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>16415</td>
<td>8207</td>
<td>.4790E-01</td>
<td>653</td>
<td>79.22</td>
<td>70.15</td>
<td>0.05</td>
<td>62.580</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>15941</td>
<td>7970</td>
<td>.6228E+00</td>
<td>269</td>
<td>66.89</td>
<td>54.90</td>
<td>0.04</td>
<td>74.030</td>
</tr>
</tbody>
</table>

RESULTS OF CONTINUOUS LB1 ALGORITHM - PROBLEM SET II, DEPTH-FIRST
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>AVG %</th>
<th>CLB1</th>
<th>ILB1</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>393</td>
<td>196</td>
<td>.7835E+00</td>
<td>1</td>
<td>15.78</td>
<td>0.50</td>
<td>0.42</td>
<td>0.29</td>
<td>2.840</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>471</td>
<td>250</td>
<td>.5014E+00</td>
<td>81</td>
<td>11.42</td>
<td>0.50</td>
<td>0.45</td>
<td>0.22</td>
<td>3.010</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>1</td>
<td>1</td>
<td>.5856E+01</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.730</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>1515</td>
<td>757</td>
<td>.8929E+00</td>
<td>353</td>
<td>13.20</td>
<td>0.50</td>
<td>0.11</td>
<td>0.33</td>
<td>11.390</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>6383</td>
<td>3197</td>
<td>.6817E-02</td>
<td>217</td>
<td>41.17</td>
<td>0.50</td>
<td>0.09</td>
<td>0.25</td>
<td>40.170</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>1511</td>
<td>760</td>
<td>.2387E+01</td>
<td>195</td>
<td>15.45</td>
<td>0.50</td>
<td>0.18</td>
<td>0.28</td>
<td>9.710</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>7287</td>
<td>3643</td>
<td>.1259E+00</td>
<td>167</td>
<td>28.47</td>
<td>0.50</td>
<td>0.01</td>
<td>0.14</td>
<td>46.400</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>13399</td>
<td>6699</td>
<td>.4790E-01</td>
<td>819</td>
<td>34.71</td>
<td>0.50</td>
<td>0.04</td>
<td>0.15</td>
<td>74.270</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>12303</td>
<td>6151</td>
<td>.6228E+00</td>
<td>143</td>
<td>24.93</td>
<td>0.50</td>
<td>0.02</td>
<td>0.18</td>
<td>81.840</td>
</tr>
</tbody>
</table>

RESULTS OF INTEGER LB1 ALGORITHM - PROBLEM SET II, DEPTH-FIRST
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>50X10</td>
<td>0.50</td>
<td>991</td>
<td>495</td>
<td>7835E+00</td>
<td>1</td>
<td>1.60</td>
<td>0.0</td>
<td>0.41</td>
<td>0.09</td>
<td>37.280</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>30X10</td>
<td>0.50</td>
<td>281</td>
<td>145</td>
<td>5014E+00</td>
<td>51</td>
<td>12.61</td>
<td>23.79</td>
<td>0.29</td>
<td>0.45</td>
<td>8.840</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>40X12</td>
<td>0.75</td>
<td>1</td>
<td>1</td>
<td>5856E+01</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.480</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>50X10</td>
<td>0.50</td>
<td>339</td>
<td>169</td>
<td>8929E+00</td>
<td>3</td>
<td>29.48</td>
<td>11.34</td>
<td>0.02</td>
<td>0.75</td>
<td>30.990</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>30X12</td>
<td>0.25</td>
<td>785</td>
<td>392</td>
<td>6838E-02</td>
<td>5</td>
<td>67.12</td>
<td>62.00</td>
<td>0.03</td>
<td>0.69</td>
<td>50.810</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>40X10</td>
<td>0.65</td>
<td>45</td>
<td>30</td>
<td>2402E+01</td>
<td>42</td>
<td>34.11</td>
<td>23.07</td>
<td>0.38</td>
<td>0.46</td>
<td>5.520</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>50X10</td>
<td>0.35</td>
<td>123</td>
<td>62</td>
<td>1259E+00</td>
<td>10</td>
<td>86.87</td>
<td>60.47</td>
<td>0.02</td>
<td>0.90</td>
<td>27.820</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>30X12</td>
<td>0.35</td>
<td>91</td>
<td>49</td>
<td>4790E-01</td>
<td>20</td>
<td>86.00</td>
<td>70.15</td>
<td>0.09</td>
<td>0.78</td>
<td>16.700</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>40X12</td>
<td>0.50</td>
<td>191</td>
<td>98</td>
<td>6228E+00</td>
<td>18</td>
<td>58.21</td>
<td>54.90</td>
<td>0.03</td>
<td>0.55</td>
<td>53.170</td>
</tr>
</tbody>
</table>

RESULTS OF LB3 ALGORITHM - PROBLEM SET II, DEPTH-FIRST
<table>
<thead>
<tr>
<th>NO.</th>
<th>RANGE</th>
<th>SIZE</th>
<th>MEAN</th>
<th>ENUM</th>
<th>STOR</th>
<th>SOLN</th>
<th>NODE</th>
<th>GAP</th>
<th>OBJ</th>
<th>FATH</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>15X10</td>
<td>0.50</td>
<td>1567</td>
<td>783</td>
<td>.2260E+00</td>
<td>883</td>
<td>34.68</td>
<td>3.85</td>
<td>0.24</td>
<td>2.160</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>15X10</td>
<td>0.50</td>
<td>1019</td>
<td>509</td>
<td>.2238E+00</td>
<td>845</td>
<td>19.98</td>
<td>10.42</td>
<td>0.31</td>
<td>1.410</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>20X12</td>
<td>0.75</td>
<td>3647</td>
<td>1823</td>
<td>.2957E+01</td>
<td>2171</td>
<td>14.17</td>
<td>10.87</td>
<td>0.22</td>
<td>7.090</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>15X10</td>
<td>0.50</td>
<td>1895</td>
<td>947</td>
<td>.2328E+00</td>
<td>633</td>
<td>46.83</td>
<td>28.14</td>
<td>0.20</td>
<td>2.820</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>20X12</td>
<td>0.25</td>
<td>8757</td>
<td>4378</td>
<td>.2038E-02</td>
<td>2349</td>
<td>85.25</td>
<td>52.78</td>
<td>0.17</td>
<td>17.930</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>15X10</td>
<td>0.65</td>
<td>3965</td>
<td>1982</td>
<td>.1028E+01</td>
<td>2859</td>
<td>19.36</td>
<td>3.75</td>
<td>0.19</td>
<td>6.010</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>10X10</td>
<td>0.35</td>
<td>6569</td>
<td>3284</td>
<td>.7741E-02</td>
<td>5679</td>
<td>87.14</td>
<td>65.72</td>
<td>0.07</td>
<td>7.580</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>20X12</td>
<td>0.35</td>
<td>15137</td>
<td>7568</td>
<td>.3203E-01</td>
<td>1969</td>
<td>70.68</td>
<td>61.21</td>
<td>0.05</td>
<td>32.190</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>20X12</td>
<td>0.50</td>
<td>14373</td>
<td>7186</td>
<td>.2718E+00</td>
<td>5039</td>
<td>66.12</td>
<td>48.96</td>
<td>0.05</td>
<td>31.010</td>
</tr>
</tbody>
</table>

RESULTS OF CONTINUOUS LB1 ALGORITHM - PROBLEM SET 1, DEPTH-FIRST
<table>
<thead>
<tr>
<th>NO.</th>
<th>PROB</th>
<th>NODES SIZE</th>
<th>OPT</th>
<th>INIT</th>
<th>COMP %</th>
<th>LB3 NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>15X10 0.50</td>
<td>1479 739</td>
<td>.2260E+00</td>
<td>849</td>
<td>34.7</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>15X10 0.50</td>
<td>949 474</td>
<td>.2238E+00</td>
<td>777</td>
<td>28.1</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
<td>20X12 0.75</td>
<td>3373 1686</td>
<td>.2957E+01</td>
<td>1991</td>
<td>36.3</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>15X10 0.50</td>
<td>1827 913</td>
<td>.2328E+00</td>
<td>615</td>
<td>68.2</td>
</tr>
<tr>
<td>5</td>
<td>0.50</td>
<td>20X12 0.25</td>
<td>8683 4341</td>
<td>.2038E-02</td>
<td>2337</td>
<td>99.5</td>
</tr>
<tr>
<td>6</td>
<td>0.70</td>
<td>15X10 0.65</td>
<td>3999 1999</td>
<td>.1028E+01</td>
<td>2899</td>
<td>65.9</td>
</tr>
<tr>
<td>7</td>
<td>0.70</td>
<td>10X10 0.35</td>
<td>6555 3277</td>
<td>.7741E-02</td>
<td>5671</td>
<td>97.4</td>
</tr>
<tr>
<td>8</td>
<td>0.70</td>
<td>20X12 0.35</td>
<td>15119 7559</td>
<td>.3203E-01</td>
<td>1969</td>
<td>99.5</td>
</tr>
<tr>
<td>9</td>
<td>0.90</td>
<td>20X12 0.50</td>
<td>14311 7155</td>
<td>.2718E+00</td>
<td>5027</td>
<td>95.1</td>
</tr>
</tbody>
</table>

RESULTS OF HYBRID ALGORITHM - PROBLEM SET I, DEPTH-FIRST
Bibliography


The vita has been removed from the scanned document