NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

SOLVING SET COVERING PROBLEMS
USING
HEURISTICS WITH BRANCH AND BOUND

BY

Kook Jin, Nam

September 1984

Thesis Advisor: R.K. Wood

Approved for public release; distribution unlimited.

U222982

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Solving Set Covering Problems Using Heuristics with Branch and Bound		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kook Jin, Nam		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE September 1984
		13. NUMBER OF PAGES 57
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Heuristic
Branch and Bound
Algorithm
Set Covering
Enumeration
Cutting Plane
Partitioning
Separation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm based on simple heuristics is presented for an important class of all-binary integer linear programs known as the set covering problem. In spite of its very special form, the set covering problem has many practical applications. Optimal solutions to problems derived from these applications are difficult to obtain using known methods.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(20. ABSTRACT Continued)
Various solution techniques are investigated based on heuristic algorithms that obtain upper and lower bounds on the optimal solution value together with branch and bound enumeration. These solution techniques are effective on some problems. Computational results are reported for several large-scale real-world problems and several artificial problems.

Approved for public release; distribution unlimited.

Solving Set Covering Problems Using Heuristics with Branch and Bound

by

Nam, Kook Jin
Major, Republic of Korea Army
B.S., Korea Military Academy, 1975

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1984

Thesis
N2393
c.1

ABSTRACT

An algorithm based on simple heuristics is presented for an important class of all-binary integer linear programs known as the set covering problem. In spite of its very special form, the set covering problem has many practical applications. Optimal solutions to problems derived from these applications are difficult to obtain using known methods. Various solution techniques are investigated based on heuristic algorithms that obtain upper and lower bounds on the optimal solution value together with branch and bound enumeration. These solution techniques are effective on some problems. Computational results are reported for several large-scale real-world problems and several artificial problems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. INTRODUCTION

Set Covering Problems (SCPs) comprise an important class of all-binary (0-1) Integer Linear Programs (ILPs). The SCP model is well-known and has many practical applications in diverse areas such as vehicle routing, facility location and capital budgeting. The set covering problem is a theoretically difficult problem in that it is NP-complete [Ref. 1]. However, there exist several methods for obtaining solutions to SCPs for quite large real-world problems. In this study, heuristics together with branch and bound enumeration are tested as a solution method for solving several large-scale SCPs.

There are several reasons for using heuristics with branch and bound instead of using cutting plane methods, LP-based branch and bound, or some other technique. First, not all researchers have access to good large-scale LP systems on which to base cutting plane or branch and bound algorithms. Any competent researcher should be able to program a heuristic-based method with a modest amount of effort. The second reason for wanting heuristic-based methods is that more complicated techniques are subject to failure as a result of degeneracy, numerical instability and slowness. For instance, the systems based on solving the LP relaxation, both cutting plane and branch and bound, fail when the LPs are difficult to solve because of their size, or because of basis structures which are hard to invert, or because the LP gives weak bounds. See [Ref. 2] and [Ref. 3].

B. THE SET COVERING PROBLEM

The SCP is an integer program of the form:

$$(1) \quad \text{MIN} \sum_{j=1}^{n} c_j x_j$$

$$(2) \quad \text{S.T.} \sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad i = 1,\ldots,m$$

$$(3) \quad x_j \text{ binary} \qquad j = 1,\ldots,n$$

where each $a_{ij} = 0$ or 1

$b_j > 0$ and integer

$c_j \geq 0.$

A minimal cost set of columns must be selected from the coefficient matrix A such that the right-hand side b is covered or satisfied. Typically, right-hand-side values are all 1s. Closely related to the SCP is the set partitioning problem(SPP) where (2) is replaced by (4).

$$(4) \quad \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1,\ldots,m.$$

The SPP is the same as the SCP except that each row i must be covered exactly b_i times instead of at least b_i times.

C. USES OF SCP AND SPP

Set covering problems and set partitioning problems have been studied widely because of their many practical applications and simple binary structure. Bausch [Ref. 2] and Balas and Padberg [Ref. 4] give a large collection of references to applications which are given below for completeness along with some more recent references. Only references 6,7,9,16 and 25 are unsighted.

1.	Airline Crew Scheduling	[Ref. 5],[Ref. 6],[Ref. 7] [Ref. 8],[Ref. 9],[Ref. 10]
2.	Airline Fleet Scheduling	[Ref. 11]
3.	Truck Deliveries	[Ref. 12],[Ref. 13] [Ref. 14],[Ref. 15] [Ref. 16]
4.	Political Districting	[Ref. 17],[Ref. 18]
5.	Information Retrieval	[Ref. 19]
6.	Symbolic Logic	[Ref. 20]
7.	Switching Theory	[Ref. 21],[Ref. 22] [Ref. 23],[Ref. 24]
8.	Stock Cutting	[Ref. 25]
9.	Line Balancing	[Ref. 26]
10.	Capacity Balancing	[Ref. 27]
11.	PERT-CPM	[Ref. 20]
12.	List Selection	[Ref. 28]
13.	Tanker Routing	[Ref. 29]
14.	Frequency Allocation	[Ref. 30]
15.	Tracking Problems	[Ref. 31]
16.	Vehicle Routing	[Ref. 32]
17.	Sales Territory Design	[Ref. 33]
18.	Coloring Problem	[Ref. 34]
19.	Cyclic Scheduling Problem	[Ref. 35],[Ref. 36]
20.	Disconnecting Paths in a Graph	[Ref. 37],[Ref. 38]
21.	Capital Investment	[Ref. 39]
22.	Location of Offshore Drilling Platforms	[Ref. 40]
23.	Facilities Location	[Ref. 41]

A truck routing problem will be described here for the purpose of illustrating both the SCP and the SPP. The SCP example is described first. The headquarters of the First Corps of the Republic of Korea Army has 8 divisions to supply using 7 possible delivery routes. It is assumed that the cost of each route is measured in dollars here, but costs could also be measured in time, miles travelled, trucks used, etc. The incidence matrix A which is shown in Table 1 consists of 1s and 0s such that

$$a_{ij} = \begin{cases} 1 & \text{if route j goes through division i} \\ 0 & \text{otherwise.} \end{cases}$$

TABLE 1

Truck Routing Example

Point	R1	R2	R3	R4	R5	R6	R7
#1	1	0	1	1	0	0	0
#2	1	0	0	1	0	0	0
#3	1	1	0	1	0	0	0
#4	1	1	0	0	0	1	0
#5	0	0	1	0	1	1	0
#6	0	1	1	0	1	0	0
#7	0	0	1	1	1	0	0
#8	0	0	1	1	0	0	1
Costs	7	8	10	12	6	5	5

A set of truck routes of minimal cost is to be determined in such a way that at least one truck route should go through each supply point.

This problem is an SCP

where $A = \{ a_{ij} \}$

$\underline{b} = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)^T$ and

$\underline{c} = (7\ 8\ 10\ 12\ 6\ 5\ 5)$.

Variable x_j has the value 1 if truck route j is in the minimum cost set of routes and it has value 0 otherwise. The optimal solution is

$$\underline{x} = (1, 0, 1, 0, 0, 0, 0)^T,$$

with objective value 17. The solution of the above problem using branch and bound will be demonstrated in Chapter 2.

Suppose, on the other hand, that trucks are picking up supplies and that the net cost of route j is given by

c_j = cost of route j - value of supplies at all points in route j. With this cost structure, no overcovering of any row may be allowed and this problem becomes an SPP. The optimal solution for the SPP is

$$\underline{x} = (1, 0, 0, 0, 1, 0, 1)^T,$$

with objective value 18. Since this SPP is a restriction of the previous SCP, it is to be expected that the optimal solution to the SPP will be no better than the optimal solution to the SCP.

D. SOLUTION METHODS FOR THE SCP

1. Cutting Planes

One method of solving a general ILP is by attempting to define the optimal integer solution of the ILP as an extreme point of a convex polyhedron generated by the original linear constraints plus some additional constraints called "cuts." The technique is applied to the SCP by first solving the linear relaxation:

$$\text{(5)} \qquad \begin{aligned} &\text{MIN} \quad \underline{cx} \\ &\text{S.T.} \quad A\underline{x} \ge \underline{b} \\ &\qquad\quad \underline{x} \ge \underline{0}. \end{aligned}$$

Solve this relaxation. If the solution is integer, then the solution must be optimal. Otherwise, derive a valid cut, i.e., a linear constraint which is satisfied by all integer solutions to SCP, but which is violated by the current non-integer solution. Add this contraint to the problem and solve the new restricted problem. Continue solving the restricted linear programs and adding cuts until either an integer solution is obtained or numerical difficulties force a halt to the process.

2. Branch and Bound

Branch and bound is an optimization technique that uses a tree search enumeration approach to the solution of a general ILP:

$$(6) \quad \begin{array}{ll} \text{MIN} & \underline{c}\underline{x} \\ \text{S.T.} & A\underline{x} \geq \underline{b} \\ & \underline{x} \geq \underline{0} \\ & \underline{x} \text{ integer.} \end{array}$$

Following Garfinkel [Ref. 41], denote the set of feasible solutions to (6) by

$$S = \{ \underline{x} \mid A\underline{x} \geq \underline{b},\ \underline{x} \geq \underline{0},\ \underline{x} \text{ integer} \}$$

Instead of attempting to solve directly over S, the set is successively divided into smaller sets which have the property that any optimal solution must be in at least one of the sets. This is called separation and is often illustrated by an enumeration tree with its root node at the top of the tree and with restricted subproblems below the root (See Figure 2.1). Each node of the enumeration tree corresponds to a subproblem of (6). That is, node k is the problem

$$(7) \quad \text{MIN} \quad \underline{c}\underline{x}\ , \quad \underline{x} \text{ is in } S_k$$

where S_k is a subset of S.

In a binary ILP, S_k is S with additional constraints which fix certain variables to 0 or 1. As the enumeration proceeds further down the enumeration tree, the subsets become progressively smaller until it finally becomes possible to solve (7) exactly or at least to determine whether or not it contains a potentially optimal solution. Subproblems are discarded or "fathomed" when (7) is solved or when it is determined that a subset cannot contain a solution better than the best known solution to (6). Upper and lower bounds on the optimal solution are calculated for each subproblem allowing for more efficient fathoming of nodes. The success or failure of branch and bound is largely dependent on the accuracy of these bounds.

Branch and bound algorithms are often primal in the sense that they proceed from one feasible solution to another until optimality is verified. In fact they may find optimal or near optimal solutions at an early stage in the enumeration process and spend the majority of the time verifying optimality by improving bounds.

E. IMPLEMENTATION

1. Introduction

Most large-scale mathematical programming problems have special structure which is exploited in the implementation of mathematical programming solvers. Examples of such special structure are sparsity of the constraint matrix and the frequent occurrence of some coefficient values. To take advantage of this structure, the computer programs written for this study are written as subroutines embedded in a large-scale optimization test bed called the X System or simply XS [Ref. 42]. XS is designed to solve linear programming problems, 0-1 programming problems, nonlinear programming problems and mixed 0-1/linear/nonlinear programming problems. XS uses sparse matrix techniques common to many mathematical programming systems. A more specialized system using binary vectors to represent the A matrix [Ref. 43] might be faster for some problems but less flexible.

2. Input Data Format

In this study, to make data manipulation easy and convenient, the data format described by Bausch [Ref. 2] is used since this format has many advantages for large-scale problems. The advantages are as follows.

a. It is compact.

b. Storage requirements are easily calculated.

c. Data generation problems are simplified.

d. Column manipulation of data input is made easy since all information for each column is contiguous.

e. This column format is easily generated by commercially available (MPS) problem generation systems.

The data input format consists of three sets of card images:

a. Problem dimensions. Format (3I6) (One Card)

M = Number of rows

N = Number of columns

NZEL = Number of non-zero elements.

b. Constraint ranges. Format (2A4, 2E16.8) (M Cards)

IR = Row index i

RL = Lower range limit b_i

RU = Upper range limit (always ∞).

c. Column Data. (N or More Cards) (2A4, F14.3, 10I5)

JC = Column index j

C = Column cost coefficient c_j

NCE = Number of non-zero elements in the column

IR row addresses of non-zero coefficients.

If NCE is greater than 9, additional column cards are needed to hold the row addresses for that column. The format for additional column cards is (20X, 10I5).

An example of this data format is shown in Appendix A for the truck routing example of Table 1.

3. Test Problems

Eight test problems are evaluated in this thesis. These problems consist of four real-world problems (American, Bus, Tiger and Truck), and four artificial problems (Steiner1, Steiner2, Steinr1a and Steinr2a). Steiner1 and Steiner2 are problems devised in [Ref. 49] and are guaranteed to require extensive enumeration when using LP-based branch and bound since the LP bounds are so weak. Steinr1a and Steinr2a are Steiner1 and Steiner2 transposed, respectively.

Some of the problems are, in fact, pure SPPs. However, we have converted these problems into SCPs reasoning that the derived SCP should still be representative of a true SCP. The characteristics of these problems are shown in Table 2 where NZEL is the total number of nonzeros in the constraint matrix and NCE is the average number of nonzeros in each column. All these problems except Truck are typical set covering problems which have right-hand sides equal to 1. The Truck problem has a general right-hand side. All of these test problems were run on an IBM3033 under VM/CMS. Computation times reported in the following chapters are accurate to the number of decimal places shown.

TABLE 2

Problem Dimensions

Problem	Rows	Columns	NZEL	NCE	Model
American	95	9318	57293	6.0	SPP
Bus	56	530	3339	6.3	SPP
Steiner1	117	27	352	13.0	SCP
Steiner1A	27	117	351	13.0	SCP
Steiner2	330	45	991	22.0	SCP
Steiner2A	45	330	991	22.0	SCP
Tiger	160	636	4134	6.7	SPP
Truck	239	4752	30075	8.0	SCP

II. BRANCH AND BOUND ENUMERATION

A. INTRODUCTION

In this chapter we introduce "branch and bound" enumeration which will be used in chapter 5 to solve SCPs. Branch and bound is an optimization technique that uses tree enumeration together with upper bounds and lower bounds on the objective function. These bounds help to accelerate the fathoming process and reduce enumeration. In this chapter, we describe branch and bound in terms of a minimizing binary ILP, and discuss the importance of good bounds and good branching strategies. In a binary ILP, a separation is effected by fixing a binary variable to its possible values, 0 and 1. Thus, every separation of a problem is, in fact, a partition of the problem into two subproblems.

The discussion of branch and bound is limited to a "depth-first" search or exploration of the enumeration tree since this is the method that was used in this research. More general techniques are possible (See Garfinkel and Nemhauser [Ref. 43].) but these all require substantially more storage and general overhead. Most commercial branch and bound packages utilize a depth-first search. Depth-first search simply means that when a separation is defined, one of the nodes created by the separation is immediately selected to be the next subproblem, and when a node is fathomed, the enumeration always backtracks to the most recently created live node which is the "twin" of a node already explored.

There are two important parts of any branch and bound algorithm. First, good upper and lower bounds must be obtained on the optimal solution. The closer the bounds are to the optimal solution, the fewer nodes (subproblems) must be enumerated. There exist several ways of obtaining upper and lower bounds on an SCP. Methods for obtaining computationally simple lower bounds will be described in Chapter 4. Upper bounds on the optimal solution are given by heuristically obtained feasible solutions to the ILP. Heuristic solution methods for the SCP will be described in detail in the next chapter.

The second important part of the branch and bound algorithm is the method of determining which variable to fix first when a separation is defined at a node in the tree and whether that variable should be fixed to 0 or 1. This selection process is called a "branching strategy." Branching strategies are dependent on the methods being used for obtaining upper and lower bounds and the actual strategies to be used will be discussed in chapter 5. It seems obvious that if a good guess can be made as to which variables must be in the optimal solution, then fixing one of those variables first to 1 would be a good branching strategy. Of course, guessing is very difficult; otherwise we would have guessed the solution to the whole problem. Another likely strategy may be to select the least attractive variable in the incumbent and set that variable to 0. Unfortunately, as will be seen in chapter 5, no single rule seems to work well on all problems and a certain amount of case-by-case experimentation is necessary.

B. FATHOMING CRITERION

To accelerate the enumeration process and save computing time we need a criterion to decide whether or not a subproblem should be discarded at a certain point of the algorithm. Suppose that several steps of the enumeration have already been performed and that a subproblem at a particular node in the tree is being considered. Let BEST denote the smallest feasible objective value found thus far in the enumeration. Clearly, BEST is an upper bound on the optimal solution to the ILP. The feasible solution corresponding to BEST is called the "incumbent."

Now, let CLBND denote a lower bound on the optimal solution to the ILP given the restrictions at the current node. CLBND is defined to be infinity if no feasible solution to the ILP can be found given the current restrictions. Let CUBND denote a upper bound on the optimal objective value corresponding to a feasible solution to the ILP given the current restrictions. If BEST > CUBND, let BEST = CUBND and let the corresponding solution be the new incumbent. Now, the efficiency of branch and bound enumeration is based on the fact that explicit enumeration need not be extended below the current node if the "fathoming criterion" is met:

Fathoming criterion: CLBND ≥ BEST.

For problems with integer costs, fractional values for CLBND should be rounded up to the nearest integer.

From a computational viewpoint, it is useful to split the above test into two tests, however. First, compute CLBND and test if CLBND ≥ BEST; if it is, the node is fathomed. If not, only then compute CUBND, update BEST if appropriate and repeat the test. This avoids some unnecessary computation of upper bounds.

C. SEPARATION AND BRANCHING

"Branching" describes the process whereby an unexplored subproblem is selected for exploration, i.e., upper and lower bounds are computed for the node and the node is either fathomed or separated. "Separation" is the process whereby the current subproblem is separated into two or more subsubproblems, at least one of which must contain the optimal solution to the current subproblem if such a solution exists.

In a binary LP using depth-first search, branching and separation are intertwined. A separation is always a partition based on fixing a specific variable to 0 or to 1. After a separation one of the live nodes just created must be immediately selected for branching. If a node is fathomed, the most recently created live node must be selected for exploration.

D. ALGORITHM AND EXAMPLE

1. Algorithm

The following branch and bound algorithm uses depth-first exploration of the enumeration tree. The logic is exactly that used in the programs written for this thesis.

Algorithm: Depth-First Branch and Bound

STEP 0. (Initialization)
Let BEST $=\infty$, STACK = ϕ.

STEP 1. Compute CUBND given restrictions defined by STACK.
If CLBND $\geq$ BEST, go to step 5.

STEP 2. Compute CUBND given restrictions defined by STACK.
If CUBND < BEST, then let BEST = CUBND and save incumbent.
If CLBND $\geq$ BEST, go to step 5.

STEP 3. (Branching) Select an unfixed variable j to fix and determine whether to fix it to 0 or 1.

STEP 4. Put vertex j in STACK with information indicating whether it is fixed to 0 or 1 and that its twin has not yet been explored.
Go to step 1.

STEP 5. (Backtrack)

If STACK = ϕ, then go to step 7.

STEP 6. Remove j from top of STACK.

If its twin has been explored, go to step 5.

Otherwise, put j back on STACK fixing j to the complement of its previous value and noting that its twin has already been explored.

Go to step 1.

STEP 7. Termination

If BEST $=\infty$, there exists no feasible solution.

Otherwise, current incumbent is optimal.

End of Algorithm: Branch and Bound

2. <u>Example</u>

The example below illustrates the above algorithm on the SCP defined in Table 1. Lower bounds on the solutions at each node are obtained by finding a feasible solution to the dual of the LP relaxation of the SCP (See section B in Chapter 4.). The upper bounds are obtained by using an "addition heuristic" which successively adds columns to a partial cover until a complete cover is obtained (See section C in Chapter 3.). Separation is effected by randomly selecting a variable x_j among all variables not in the current solution obtained by the addition heuristic. The branch corresponding to $x_j = 1$ is explored first. The enumeration tree is shown in Figure 2.1.

a. Initialize: BEST $= \infty$

b. Node 0: CLBND = 17.0

CLBND < BEST so continue.

CUBND = 18.0, $\underline{X}_0$ = (1,0,0,0,1,0,1)

CUBND < BEST so let BEST = CUBND and $\underline{X}_0$ becomes the incumbent.

CLBND < BEST so variable x_2 is selected

for branching. Fix variable x_2 to 1 first.

c. Node 1: Given that x_2 = 1, CLBND = 25.0.

Since CLBND ≥ BEST, backtrack to the twin of this node which has not been explored.

d. Node 2: Given that x_2 = 0, CLBND = 17.0.

CLBND < BEST so continue.
CUBND = 18.0. No improvement over incumbent.
Since CLBND < BEST, select x_5 for branching.

Set x_5 = 1.

e. Node 3: Given that x_2 = 0 and x_5 = 1, CLBND = 18.0.

Since CLBND ≥ BEST, backtrack to the twin of this node which has not been explored.

f. Node 4: Given that x_2 = 0 and x_5 = 0, CLBND = 17.0.

Since CLBND < BEST, continue.
CUBND = 17.0 for $\underline{X}_4$ = (1,0,1,0,0,0,0).

Since CUBND < BEST, let BEST = CUBND and let $\underline{X}_4$ be the new incumbent.

Since CLBND ≥ BEST, backtrack.
No live nodes exist, so the current incumbent $\underline{X}_4$ is optimal with objective value 17.0.

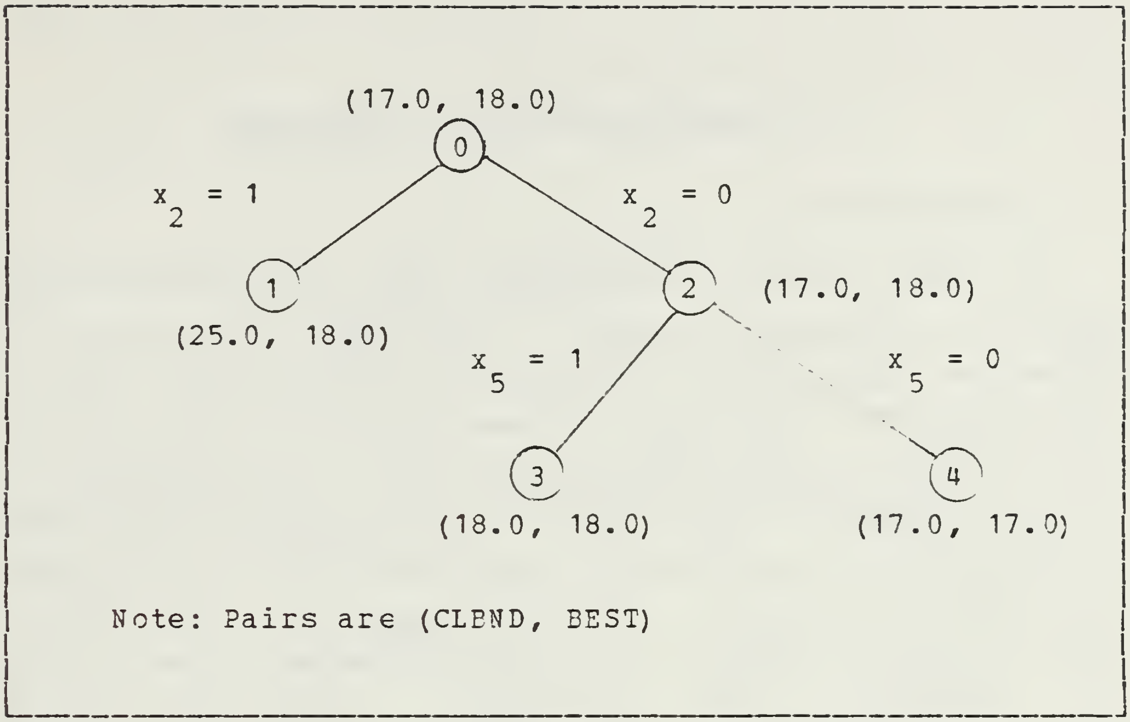

Figure 2.1 Enumeration Tree for Truck Routing Example.

III. HEURISTIC SOLUTION TECHNIQUES

A. INTRODUCTION

Two basic heuristic techniques exist for obtaining good feasible solutions to SCPs: "addition" heuristics and "deletion" heuristics. These two heuristics are used in this study for the purpose of generating solution sets and upper bounds. These heuristics are not guaranteed to solve the SCP optimally but can be used to get good upper bounds on the optimal solutions which are essential in the branch and bound enumeration. Feasible solutions to the SCP are easily obtained because of the SCP's greater than or equal to constraints and nonnegative constraint matrix. Computational results are given in section D.

B. ADDITION HEURISTIC

An addition heuristic begins with the infeasible solution $\underline{x}=\underline{0}$ and successively sets to 1 that variable x_j which myopically minimizes effective cost. The effective cost associated with x_j is c_j/p_j, where p_j is a penalty which in some way reflects the amounts of infeasibility currently being contributed by $x_j = 0$. The addition heuristic can be stopped when a feasible solution is obtained but it is possible that the cover produced is not minimal and a second phase should be added which deletes any columns in the cover which are redundant.

Algorithm: Addition Heuristic

Input: The SCP matrix and vectors A, $\underline{c}$ and $\underline{b}$.

Output: Upper bound to SCP solution.

STEP 0. "Initialization"

$I = \{1,2,\ldots,m\}$, $J = \{1,2,\ldots,n\}$,

$J' = \phi$, $\underline{b}' = \underline{b}$, CUBND = 0.

For each column $j \in J$

Compute a penalty $p_j > 0$.

Let h_j = number of nonzeros in column j.

STEP 1. If $\underline{b}' < \underline{0}$ or J is empty, go to step 3.

Otherwise, let

$$j_0 = \operatorname*{argmin}_{j \in J} \frac{c_j}{p_j}$$

$J = J - j_0$

$J' = J' + j_0$

$\text{CUBND} = \text{CUBND} + c_{j_0}$.

STEP 2. For each i such that $a_{ij_0} = 1$

Let $b'_i = b'_i - 1$.

If $b'_i = 0$ (update column sums)

For each j such that $a_{ij} = a_{ij_0} = 1$,

Let $h_j = h_j - 1$.

If $h_j = 0$, let $J = J - j$.

If $\underline{b}' \le \underline{0}$, go to STEP 3.

Otherwise,

For each column $j \in J$

Update penalties p_j if necessary.

Go to step 1.

STEP 3. "Generating minimal cover"

For each each $j \in J'$

If column j is redundant

Let CUBND = CUBND - c_j.

Let $J' = J' - j$.

STEP 4. "Termination"

Halt. If J' is a cover, CUBND is an upper bound on the SCP.

Otherwise, no feasible solution exists.

End of Addition Heuristic

Two different penalty functions have been tested with the above addition heuristic: $p_j = h_j$ and $p_j = k_j$ where k_j is the initial column sum (number of nonzeros in the column) which is never updated. Kovac [Ref. 44] suggests using k_j as part of a heuristic for obtaining both upper and lower bounds on the optimal solution to an SCP. The results obtained using this penalty are not reported here, however, since they are so poor. It should be noted that more complicated penalty functions could certainly be defined such as $p_j = \log(h_j)$. In addition, instead of selecting the minimum c_j/p_j the minimum of a more general functional form $g(c_j, p_j)$ could be selected.

C. DELETION HEURISTIC

A deletion heuristic begins with the feasible solution $\underline{x}=\underline{1}$ and successively sets to 0 that variable x_j which myopically minimizes effective profit c_j/p_j. Here p_j is some penalty reflecting the amount of overcovering which column j is contributing. The deletion heuristic stops when no additional variables can be set to 0 without creating an infeasiblility implying that the cover obtained is minimal. The following algorithm carries out the above ideas.

Algorithm: Deletion Heuristic

Input: The SCP matrix and vectors A, $\underline{c}$ and $\underline{b}$.

Output: Upper bound to SCP.

STEP 0. "Initialization"

$I = \{1,2,...,m\}$, $J = \{1,2,...,n\}$,

$$J' = \phi, \quad \underline{b}' = \underline{b}, \quad CUBND = \sum_{j=1}^{n} c_j$$

For each $i \in I$,

let h_i = number of nonzeros in row i.

For each $j \in J$,

let h_j = number of nonzeros in column j.

compute a penalty $p_j > 0$.

STEP 1. If $\underline{b}' < \underline{0}$ or J is empty, go to STEP 4.

$$j_0 = \underset{j \in J}{\operatorname{argmax}} \frac{c_j}{p_j}$$

STEP 2. For each i such that $a_{ij_0} = 1$,

If $b'_i = h_i$, go to STEP 4.

STEP 3. $J = J - j_0$

$CUBND = CUBND - c_{j_0}$

For each row j such that $a_{ij} = 1$.

let $h_i = h_i - 1$.

Go to STEP 1.

STEP 4. For each $a_{ij_0} = 1$ let $J' = J' + j$

For each $a_{ij} = 1$ let $b'_i = b'_i - 1$

If $b_i = 0$, update column sum h_j letting

$J = J - j$ for any $h_j = 0$.

If $\underline{b}' < \underline{0}$, go to STEP 6.

STEP 5. For each column $j \in J$.

Update penalties p_j if necessary.

Go to STEP 1.

STEP 6. "Termination"

Halt. If J' is cover, CUBND is an upper bound on the SCP.

Otherwise, no feasible solution exists.

End of Algorithm Deletion Heuristic

The deletion heuristic has only been tested using $p_j = h_j$. The comments on the possible use of more general functional forms in the addition heuristic are also valid here, but have not been tested.

D. COMPUTATIONAL RESULTS

An addition heuristic and a deletion heuristic have been coded for purposes of comparison. The results are summarized in Table 3. As can be seen, the addition heuristic is faster than deletion heuristic, but the upper bound from the addition heuristic is not as good as that obtained by the deletion heuristic except for problems Steiner1A and Truck. Although these results tend to indicate that the deletion heuristic is better, the true test of usefulness in solving SCPs will have to wait until chapter 5 where the heuristics are embedded in a branch and bound algorithm.

TABLE 3

Computational Results of Upper Bounds

Problem	Addition		Deletion	
	Value	Time	Value	Time
American	3.532	0.35	3.364	35.13
Bus	5.253	0.03	5.192	0.64
Steiner1	19.0	0.00	19.0	0.00
Steinr1a	9.0	0.00	10.0	0.00
Steiner2	32.0	0.00	30.0	0.01
Steinr2a	16.0	0.00	16.0	0.24
Tiger	59.264	0.11	59.173	0.92
Truck	367.64	1.19	389.62	53.35

The above heuristic techniques are one-pass methods and are the only methods implemented in this research. Multiple-pass methods exist and are mentioned here for completeness. The most straightforward multiple-pass method is called the "1-opt" method [Ref. 51]. This method first uses an addition or deletion heuristic to obtain a minimal cover. Then, each column in the current solution is checked against the columns not in the solution to determine if a one-for-one exchange can be made which maintains a feasible cover while reducing total cost. The 1-opt method is an example of an "exchange heuristic." The basic idea can be extended to a k-way exchange resulting in the "k-opt" method.

IV. LOWER BOUNDS ON THE SCP

A. INTRODUCTION

Several methods of finding lower bounds for solutions to the SCP are described in this chapter. Getting good lower bounds on the optimal solution to an SCP is critical if optimal solutions are to be obtained using branch and bound. Lower bounds are also necessary if the accuracy of nonoptimal solutions is to be bounded when branch and bound fails to find the optimal solution. There are many possible methods of obtaining lower bounds on the SCP, all based on solving some relaxation of the the associated ILP. Several methods have been coded and are compared to decide which method should be employed within the branch and bound enumeration. Although these bounds have not been used for solving any SPPs, it should be noted that they are all valid for the SPP since the SCP is a relaxation of the SPP.

A feasible solution to the dual of the LP relaxation of the SCP provides one easily obtainable lower bound. Also, a column partitioning method is given in which the SCP is partitioned into small SCPs which can be solved exactly and whose solution values may be summed to give a lower bound. Another lower bound which is tested is the sum of all of the minimal row covering fractions. All these methods are coded and computational results compared in section E.

B. DUAL LP RELAXATION LOWER BOUND

One obvious relaxation of the SCP which can be solved to obtain a lower bound is the linear programming (LP) relaxation. This technique has been used successfully in many cases [Ref. 45]. But a problem with the LP lower bound is that it may be difficult to solve the LPs associated with many SCPs. (See for example Bausch [Ref. 2] and Salkin and Koncal [Ref. 3].) This is true because the LP may be quite large, highly degenerate and have a basis structure which is numerically unstable. However, it is possible to get a quick lower bound on the SCP by just finding a feasible solution to the dual of the LP relaxation of the SCP since,

$$w(LP_D) \leq w^*(LP_D) = v^*(LP_P) \leq v^*(SCP)$$

where $w(LP_D)$ = objective value for a feasible solution to the dual of the LP relaxation,

$w^*(LP_D)$ = optimal value of the dual LP relaxation,

$v^*(LP_P)$ = optimal value of primal LP relaxation, and

$v^*(SCP)$ = optimal value of the SCP.

Letting $\underline{1}$ denote a row vector of ones, the dual to the LP relaxation of the SCP is

$$\begin{aligned} \max \quad & \underline{b}^T \underline{u} - \underline{1}\underline{v} \\ \text{s.t.} \quad & A^T \underline{u} - \underline{I}\underline{v} \leq \underline{c} \\ & \underline{u} \geq \underline{0}, \ \underline{v} \geq \underline{0}. \end{aligned}$$

The dual LP looks very similar to the SCP itself if the dual variables $\underline{v}$ are set to zero, and a simple method for obtaining a feasible solution can be devised in a way that is similar to the greedy addition heuristic for the SCP.

Algorithm: Dual LP Relaxation Lower Bound (DLPRLB)

Input: SCP coefficient matrix and vectors A, $\underline{c}$ and $\underline{b}$.

Output: Lower bound CLBND to the SCP

STEP 0. "Initialization"

CLBND = 0

$I = \{1,2,\ldots,m\}$, $J = \{1,2,\ldots,n\}$

For each $i \in I$, let h_i = number of nonzeros in row i.

STEP 1. If I is empty, go to STEP 5. Else, let

$$i_0 = \underset{i \in I}{\text{argmax}} \frac{b_i}{h_i}$$

$$I = I - i_0$$

STEP 2. $c_{j_0} = \min_{j: a_{i_0 j} = 1} c_j$

$$J = J - j_0$$

STEP 3. CLBND = CLBND + $c_{j_0} b_i$

STEP 4. For each row i such that $a_{ij_0} = 1$,

For each column j such that $a_{ij} = 1$,

Update cost coefficients

$$c_j = c_j - c_{j_0}$$

Repeat STEP 1.

STEP 5. "Termination"

Halt. CLBND is a lower bound on the SCP.

End of Algorithm DLPRLB

At each iteration of the algorithm, dual variable u_i would be set to c_{j_0}. The actual values of the dual variables need not be retained, however, since the value of the dual objective function is just $b^T u$.

Hey [Ref. 45] gives a somewhat more complicated method for finding a feasible solution to the dual relaxation of the SCP and this is tested along with the method described above.

C. PARTITIONING LOWER BOUND

Marsten [Ref. 50] gives a method for finding a lower bound on the SCP by solving subproblems of the SCP defined on certain partitions of the coefficient matrix. It is easier to describe the method, however, in terms of a maximization problem.

Brown, McBride and Wood [Ref. 46] give a way to calculate an upper bound using a partition of the columns for a problem of the form:

$$\begin{aligned} &\text{MAX} \quad \underline{c}\underline{x} \\ &\text{S.T.} \quad A\underline{x} \le \underline{b} \\ &\qquad\quad \underline{x} \text{ binary.} \end{aligned}$$

They use the bound for estimating the size of the maximum embedded generalized network in an LP constraint matrix where $\underline{c} = \underline{1}$, $\underline{b} = \underline{2}$, and A is the transpose of the 0-1 incidence matrix associated with an LP constraint matrix. Their bound is found as follows.

Let A_1 and A_2 be a partition of the columns of A and let z, z_1 and z_2 be the solution of the maximization problem on A, A_1 and A_2 respectively. Then, $z \le z_1 + z_2$.

If A_1 is intelligently chosen, z_1 can be computed exactly. Then, A becomes A_2 and the partitioning scheme is recursively repeated until z_2 can be easily solved also.

The SCP can be converted to a maximization problem by substituting variables $\underline{1}-\underline{y}$ for $\underline{x}$ and multiplying the objective function by -1. Thus, it is not hard to see that a lower bound on the SCP can be obtained using the above method. Of course, the method can be applied directly without making the substitution. The partition of the columns

A_1 is created with respect to an arbitrary row i. The columns of A_1 are those columns of row i containing nonzeros. The minimum cost among those cost coefficients contributes an additive term to the lower bound. The variables included in the partition are never considered again and all rows with nonzero intersections in the partition are also never considered again.

Algorithm: Partitioning Lower Bound

Input: SCP coefficient matrix and vectors A, $\underline{c}$ and $\underline{b}$.

Output: Lower bound CLBND to the SCP

STEP 0. "Initialization"

$I = \{1,2,\ldots,m\}$, $J = \{1,2,\ldots,n\}$

For each $i \in I$,

let h_i = number of nonzeros in row i

STEP 1. If I is empty, go to step 4.

Let $i_0 = \operatorname{argmin}_{i \in I} h_i$

$I = I - i_0$

STEP 2. (Find the b_{i_0} minimum c_j in row i_0.)

Let $j_0 = \operatorname{argmin}_{j:a_{i_0 j}=1} c_j$.

Let CLBND = CLBND + c_{j_0}.

Let $c_{j_0} = \infty$

Let $b_{i_0} = b_{i_0} - 1$.

If $b_{i_0} > 0$, go to step 2.

STEP 3. For each j' such that $a_{i_0 j} = 1$,

For each i such that $a_{ij'} = 1$

Let $b_i = b_i - 1$

If $b_i = 0$, let $I = I - i$.

Go to step 1.

STEP 4. "Termination"

Halt. CIBND is a lower bound to the SCP.

End of Algorithm Partitioning Lower Bound

D. KOVAC'S LOWER BOUND

Consider the basic model of the SCP with all right-hand-side values equal to 1.

$$(8) \quad \text{MIN} \sum_{j=1}^{n} c_j x_j$$

$$\text{S.T} \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1,\ldots,m$$

$$x_j \text{ binary} \qquad j = 1,\ldots,n.$$

The following lemma [Ref. 44] provides a lower bound for the above SCP.

Lemma Denote the optimum of problem (8) by Z^* then

$$(9) \quad Z^* \geq \sum_{i=1}^{m} f_i = F$$

where f_i is the minimal covering fraction of row i:

$$f_i = \text{MIN} \{ r_j^0 \mid 0 < j \leq n, a_{ij} = 1 \}.$$

(Proof) Define the following new problem.

$$(10) \quad \text{MIN} \sum_{j=1}^{n} \sum_{k=1}^{m} r_j \, y_{jk}$$

$$\text{S.T} \sum_{j=1}^{n} \sum_{k=1}^{m} g_{ijk} \, y_{jk} \geq 1 \qquad i = 1,\ldots,m$$

$$y_{jk} \quad \text{binary} \qquad j = 1,\ldots,n, \; k = 1,\ldots,m$$

$$\text{where} \quad g_{ijk} = \begin{cases} a_{ij} & \text{if } j = k \\ 0 & \text{otherwise.} \end{cases}$$

To any feasible sclution $\underline{x}$ of the SCP, there corresponds a solution of problem (10) in such a way that the objective function values are equal. Specifically, for each $x_j=1$, $y_{jk} = 1$ for all k. On the other hand the minimum of problem (10) is obviously F. This proves the statement (9) of the lemma.

It is easy to extend the Kovac bound to problems with general right-hand-sides by using

$$F = \sum_{i=1}^{m} b_i f_i.$$

E. COMPUTATIONAL RESULTS

Computational results for the lower bounds described above are reported here for all the test problems (See Table 4. "DNR" indicates did not run.). It is not clear that the tightest lower bound for the complete problem will perform the best in the branch and bound enumeration, but some positive correlation is to be expected. The computational speed of the bound is also a consideration in branch and bound and some increase in speed may be traded for a loss in accuracy. Thus, the results given here are a guide to which bound will be most effective in the branch and bound algorithm but only testing with that algorithm can determine true effectiveness. The dual LP relaxation method (DLPRLB) appears to be the best in the problems Bus, Steiner1, Tiger and American. Actually, both DLPRLB and Hey's method do outstandingly well on Bus. Kovac's heuristic appears to be superior to the other bounds in Truck, Steinr1a, Steiner2 and Steinr2a. For Steinr1a and Steinr2a, the bounds are tight indicating that if good heuristic solutions can be obtained, the branch and bound algorithm should terminate very quickly.

TABLE 4

Computational Results of Lower Bounds

Problem	OPT	Kovac's		Partitioning		DLPRLB		Hey's	
		Value	Time	Value	Time	Value	Time	Value	Time
American	1.726	1.389	0.10	1.27	0.00	1.59	0.10	1.378	0.12
Bus	4.696	0.834	0.00	2.79	0.00	4.635	0.00	4.635	0.00
Steiner1	18.0	8.923	0.00	9.0	0.00	9.0	0.00	9.0	0.00
Steinr1a	9.0	9.0	0.00	2.0	0.00	2.0	0.00	2.0	0.00
Steiner2	30.0	14.7	0.00	14.0	0.00	14.0	0.00	14.0	0.00
Steinr2a	15.0	15.0	0.00	2.0	0.00	2.0	0.00	2.0	0.00
Tiger	52.751	35.36	0.00	30.29	0.00	50.93	0.03	50.1	0.03
Truck	?	206.5	0.31	149.1	0.00	131.0	0.12	DNR	-

V. COMPUTATIONAL EXPERIENCE AND DIFFICULTIES

A. RESULTS

All of the computational results reported in this section are for SCPs. The algorithms performed very well on some of the problems but not on the others. At first, small test problems were used to check the correctness of the algorithms. All algorithms worked well on these small problems. The algorithms were then tested on the eight problems described in chapter 1. To solve these problems, we have used five methods to see which method is more effective than the others. The descriptions of the methods follow.

Method 1: Lower bound: Dual LP relaxation.
Upper bound: Addition heuristic.
Separation: $j_0 = \text{argmin } c_j/h_j$ among all $x_j = 1$ in current solution.
Branching: $x_{j_0} = 1$ first.

Method 2: Lower bound: Dual LP relaxation.
Upper bound: Addition heuristic.
Separation: $j_0 = \text{argmax } c_j/h_j$ among all j not in current solution.
Branching: $x_j = 0$ first.

Method 3: Lower bound: Dual LP relaxation.
Upper bound: Deletion heuristic.
Separation: $\text{argmin } c_j/h_j$ among all variables j not in current solution.
Branching: $x_j = 1$ first.

Method 4: Lower bound: Dual LP relaxation.
Upper bound: Deletion heuristic.
Separation: argmax c_j/h_j among all variables
j not in current solution.
Branching: $x_j = 0$ first.

Method 5: Lower bound: Kovac's
Upper bound: Addition heuristic.
Separation: argmin c_j/h_j among all variables
j not in current solution.
Branching: $x_j = 1$ first.

As illustrated in Table 5, three problems were not solved optimally. We denote the actual percentage with respect to the optimal value as a "%OPT" and the provably optimal percentage as "%POPT." %POPT denotes the amount by which were able to prove that the best solution found varied from the optimal solution without knowing the optimal solution. This value is obtained by changing the CLBND ≥ BEST tests in the branch and bound algorithm to CLBND ≥ BEST-EPS where EPS is an allowable amount of error. If the branch and bound algorithm then halts, it follows that the incumbent solution is within 100%·(BEST+EPS)/BEST of the optimal solution.

TABLE 5

Computational Results Compared with Previous Results

	New Results					Previous Results	
Problems	OPT	%OPT	%POPT	Method	Time	%OPT	ILPtime
American	1.726	110.0	110.7	1	0.09	100	26.53
Bus	4.696	100	100	2	0.54	100	1.02
Steiner1	18.0	100	100	4	30.98	100	25.13
Steiner1A	9.0	100	100	5	0.01	100	0.91
Steiner2	30.0	100	100	4	96.74	100	527.08
Steiner2A	15.0	100	100	5	13.71	100	13.14
Tiger	52.751	101.5	105.0	1	1.0	100	1.71
Truck	?	?	140.0	5	25.74	?	?

All these problems except Truck are typical set covering problems which have right-hand-side values equal to 1. The Truck problem has the general right-hand-side form shown in Equation (2). The computational results are summarized in Table 5. These results are the best of the various solution methods tried. "Previous Results" indicate either those times reported by Bausch or the times we recorded using the methods of Bausch. Running times on Steiner2 are for 10000 nodes only; optimality was not proven in either our or Bausch's computation. Table 6 shows the comparison between the different solution techniques on each of the problems. The problems marked with * were not solved optimally within 1 minute of CPU time.

TABLE 6

Comparison of Various Solution Methods

Problem	OPT	Method 1		Method 2		Method 3		Method 4	
		Value	Time	Value	Time	Value	Time	Value	Time
American*	1.726	1.89	1.17	1.89	11.11	1.89	12.10	1.89	15.42
Bus	4.696	4.696	0.82	4.696	0.54	4.696	16.61	4.696	4.24
Steiner1	18.0	18.0	31.81	18.0	36.72	18.0	31.51	18.0	30.92
Steinr1a	9.0	9.0	30.21	9.0	32.93	9.0	33.83	9.0	39.22
Steiner2	30.0	30.0	111.69	30.0	100.38	30.0	105.11	30.0	96.74
Steinr2a	15.0	15.0	154.53	15.0	144.58	15.0	126.71	15.0	216.84
Tiger*	52.75	53.54	40.35	54.48	1.92	53.84	41.28	58.17	17.6
Truck*	?	354.5	19.91	350.7	15.61	389.6	56.42	389.1	52.47

B. EXAMPLE

One cf the results of the tests was that the deletion heuristic usually produces better feasible solutions than the addition heuristic both initially and further down the tree. This leads to the enumeration of fewer nodes with the deletion heuristic. Unfortunately, it does not lead to faster times because the deletion heuristic is so much slower than the addition heuristic. For example, using method 4 which includes the deletion heuristic, it is possible to solve Bus after exploring only 15 nodes. Using method 1 with the addition heuristic requires developing 53 nodes to solve Bus. Cn the other hand, the method using the deletion heuristic requires 4.24 seconds to solve the problem while the method using the addition heuristic requires only 0.82 seconds to solve the problem.

In order to illustrate the actual behavior of the algorithm, the enumeration for Bus is shown below for two different methods, method 2 and method 4. For these two methods, the enumeration trees are sufficiently small to be shown. The entire trees generated for Bus are displayed in Figure 5.1 and Figure 5.2. Note that for both methcds, the optimal solution is found at the second node of the enumeration tree. Most of the running time of the algorithm is spent proving optimality after the optimal solution is found.

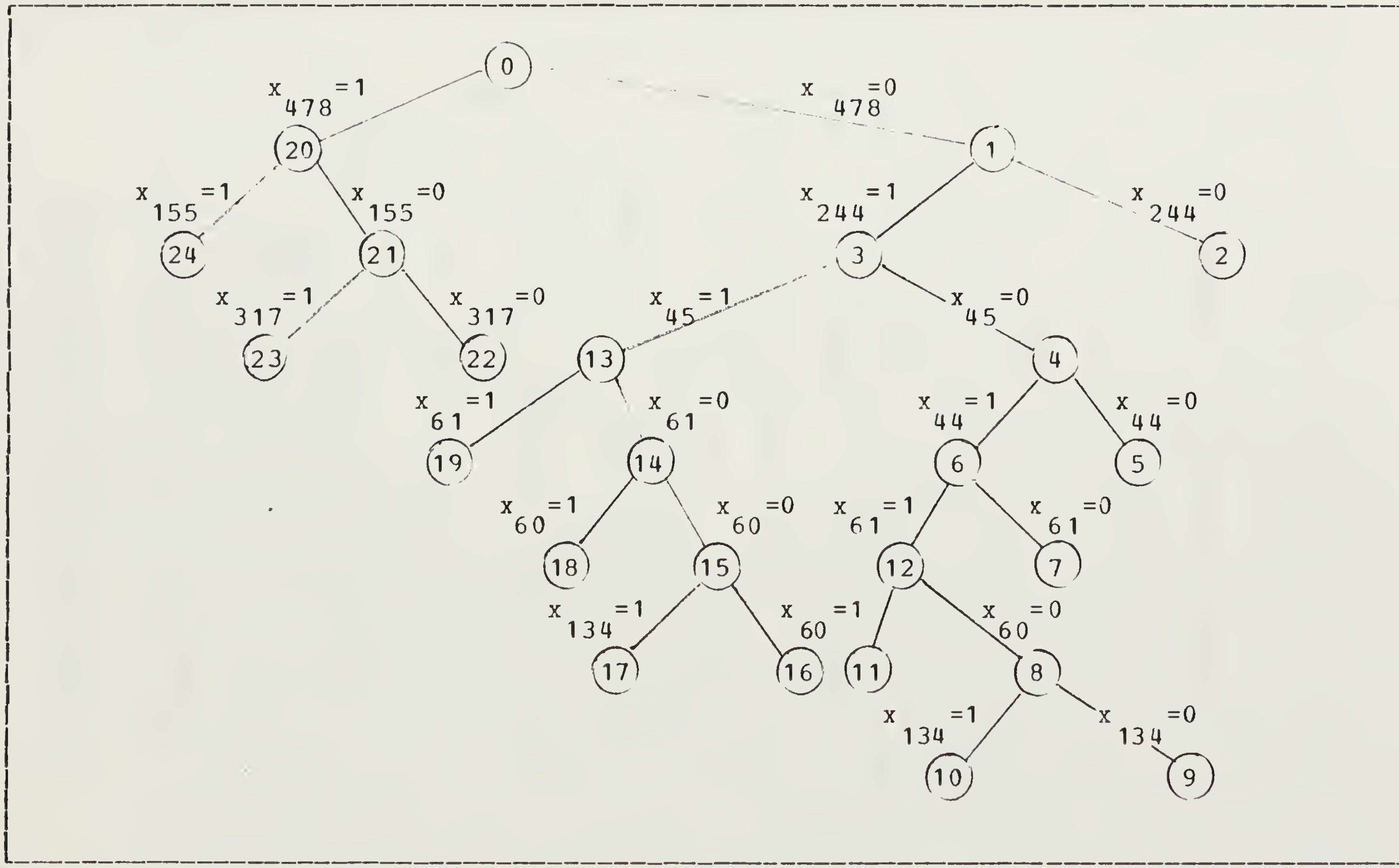

Figure 5.1 Method 2 on Bus.

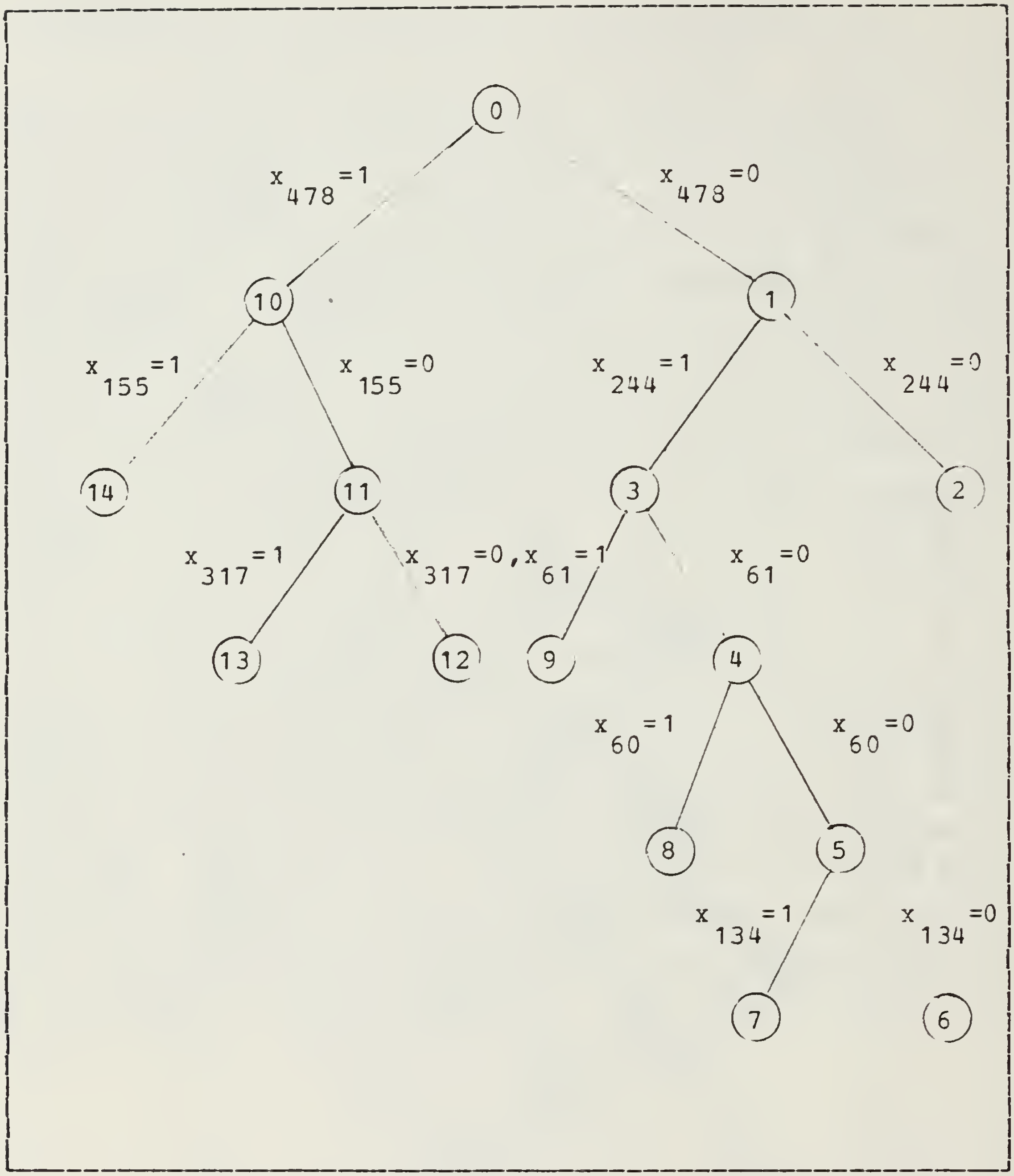

Figure 5.2 Method 4 on Bus.

VI. CONCLUSIONS AND RECOMMENDATIONS

The branch and bound enumeration method using heuristically obtained upper and lower bounds works well on some problems and poorly on others. Solution times are better than the times using the methods described by Bausch on certain problems but other problems could not be solved to optimality in a reasonable amount of time. The algorithm is largely dependent upon the quality of bounds obtained, and in certain instances these bounds are not very good.

The greedy addition heuristic used here does not perform as well as might be hoped and the deletion heuristic, which performs better, is too slow to use in most cases. Other addition heuristics should probably be tested which select that column j minimizing some function $g(c_j, h_j)$, where $g(c_j, h_j)$ is some function other than c_j/h_j such as $c_j/\log(h_j)$. In fact, Vasko and Williams [Ref. 51] have had some success selecting randomly from a number of such functions, albeit on randomly generated problems. They also utilize a 1-opt heuristic. Future research should examine the use of this and other exchange heuristics, particularly in conjunction with the addition heuristic since it may be possible to significantly improve upon the solutions obtained without sacrificing much computational speed.

The lower bound from the dual LP works quite well on some problems and poorly on others, notably Truck. Of course, the LP-based bound did not work well on Steiner1 or Steiner2 since those problems were concocted so as to have very poor LP relaxations. The high speed of computation for this lower bound does allow rapid investigation of a large number of nodes, however. In the Steiner problems, we expected that branch and bound enumeration using Kovac's lower bound might work better than the other lower bounds since the value of the initial lower bounds were stronger than the other bounds as shown in Table 4. Unfortunately, the quality of the bound does not improve rapidly enough as the enumeration proceeds. Additional research is needed to generate better heuristic solution sets and lower bounds.

APPENDIX A

DATA FORMAT FOR TRUCK ROUTING EXAMPLE

```
8       7      23
1   0.10000000D+01  0.10000000D+21
2   0.10000000D+01  0.10000000D+21
3   0.10000000D+01  0.10000000D+21
4   0.10000000D+01  0.10000000D+21
5   0.10000000D+01  0.10000000D+21
6   0.10000000D+01  0.10000000D+21
7   0.10000000D+01  0.10000000D+21
8   0.10000000D+01  0.10000000D+21
1            7.000    4    1    2    3    4
2            8.000    3    3    4    6
3           10.000    5    1    5    6    7    8
4           12.000    5    1    2    3    7    8
5            6.000    3    5    6    7
6            5.000    2    4    5
7            5.000    1    8
```

LIST OF REFERENCES

1. Garey, M. R. and Johnson, D. S., Computers and Intractability: A Guide to the Theory of NP-completeness, W. H. Freeman and Co., San Francisco, 1979.

2. Bausch, G., Computational Advances in the Solution of Large-Scale Set Covering and Set Partitioning Problems, M.S. Thesis, Naval Postgraduate School, October 1982.

3. Salkin, H. M. and Koncal, R. D., "Set Covering by an all Integer Algorithm - Computational Experience," Journal of the Association for Computing Machinery, vol. 20, pp. 189-193, 1973.

4. Balas, E. and Padberg, M. W.,"Set Partitioning: A Survey," Combinatorial Optimization, John Wiley and Sons, 1979.

5. Marsten, R. E., and Muller, M. R., "A Mixed Integer Progamming Approach to Air Cargo Fleet Planning," Management Science, vol. 26, no. 11, pp. 1096-1107, 1980.

6. Spitzer, M., "Solution to the Crew Scheduling Problem," presented at the first AGIFORS Symposium, October 1961.

7. Kolner, T. N., "Some Highlights of a Scheduling Matrix Generation System," United Airlines, presented at the Sixth AGIFORS Symposium, September 1966.

8. Arabeyre, J. P., Fearnley, J., Steiger, F. C., and Teather, W., "The Airline Crew Scheduling Problem: A Survey," Transportation Science, vol. 3, no. 2, pp. 140-163, 1969.

9. Thiriez, H., Airline Crew Scheduling: A Group Theoretic Approach, Ph.D. Dissertation, Massachusetts Institute of Technology, October 1969.

10. Marsten, R. E., Muller, M. R., and Killion, C. L., "Crew Planning at Flying Tiger: A Successful Application of Integer Programming," Management Science, vol. 25, no. 12, pp. 1175-1183, 1979.

11. Levin, A., Fleet Routing and Scheduling Problems for Air Transportation System, Ph.D. Dissertation, Massachusetts Institute of Technology, January 1969.

12. Dantzig, G. B., and Ramser, J. H., "The Truck Dispatching Problem," Management Science, vol. 6, no. 1, pp. 80-91, 1959.

13. Balinski, M. L. and Quandt, R., "On an Integer Program for a Delivery Problem," Operations Research, vol. 12, no. 8, pp. 300-304, 1964.

14. Clark, G. and Wright, S. W., "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, vol. 12, no. 4, pp. 568-581, 1964.

15. Pierce, J. F., "Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems," Management Science, vol. 15, no. 3, pp. 191-209, 1968.

16. Laskey, J. S., Optimal Scheduling of Freight Trucking, M.S. Thesis, Massachusetts Institute of Technology, June 1968.

17. Wagner, W. H., "An Application of Integer Programming to Legislative Redistricting," CROND, Inc., presented at the 34th National Meeting of ORSA, Philadelphia, Pennsylvania, November 1968.

18. Garfinkel, R. S. and Nemhauser, G. L., "Optimal Political Districting by Implicit Enumeration Techniques," Management Science, vol. 16, no. 8, pp. 495-508, 1970.

19. Day, R. H., "On Optimal Extracting From A Multiple File Data Storage System: An Application of Integer Programming," Operations Research, vol. 13, no. 3, pp. 482-494, 1965.

20. IBM Research Center Report RC-756, An Application of Linear Programmingto the Minimization of Boolean Functions, by R. Fridshal and J. H. North, June 1961.

21. Balinski, M. L., "Integer Programming: Methods, Uses, Computation," Management Science, vol. 12, no. 12, pp. 253-313, 1965.

22. IBM Research Center Report RC-756, A Statistical Study of the Minimization of Boolean Functions Using Integer, by A. Cobham, June 1961.

23. Paul, M. C. and Unger, S. H., "Minimizing the Number of States in Incompletely Specified Sequential Functions," IRE Transactions on Electronic Computers, EC-8, pp. 356-367, 1959.

24. Root, J. C., "An Application of Symbolic Logic to a Selection Problem," Operations Research, vol. 12, no. 4, pp. 519-526, 1964.

25. Pierce, J. F., "Pattern Sequencing and Matching in Stock Cutting Operations," Tappi, vol. 53, no. 4, pp. 668-678, April 1970.

26. Salveson, M. E., "The Assembly Line Balancing Problem," Journal of Industrial Engineering, vol. 6, no. 3, pp. 18-25, 1955.

27. Steinmann, H. and Schwinn, R., "Computational Experience with a Zero-One Programming Problem," Operations Research, vol. 17, no. 5, pp. 917-920, 1969.

28. Dwyer, F. R. and Evans, J. R., "A Branch and Bound Algorithm for the List Selection Problem in Direct Mail Advertising," Management Science, vol. 27, no. 6, pp. 658-667, 1981.

29. Ronen, D., Scheduling of Vessels for Shipment of Bulk and Semi-bulk Commodities Originating in a Single Area, Ph.D. Dissertation, Ohio State University, 1979.

30. Thuve, H., "Frequency Planning as a Set Partitioning Problem," European Journal of Operational Research, (Netherlands), vol. 6, no. 1, pp. 29-37, January 1981.

31. Morefield, C. L., "Application of 0-1 Integer Programming to Multitarget Tracking Problems," IEEE Trans. Autom. Control, vol. AC-22, no. 3, pp. 302-312, June 1977.

32. Cullen, F. H., Jarvis, J. J. and Ratliff, H. D., "Set Partitioning Based Heuristics for Interactive Routing," Networks, vol. 11, no. 2, pp. 125-143, 1981.

33. Shanker, R. J., Turner, R. E. and Zoltners, A. A., "Sales Territory Design: An Integrated Approach," Management Science, vol. 22, no. 3, pp. 309-320, November 1975.

34. Busacker, R. G. and Saaty, T. L., Finite Graphs and Networks, McGraw-Hill, 1965.

35. Bartholdi, III, J. J., Orlin, J. B. and Ratliff, H. D., "Cyclic Scheduling Via Integer Programs with Circular Ones," Operations Research, vol. 28, no. 5, pp. 1074-1085, 1980.

36. Bartholdi, III, J. J., "A Guaranteed-Accuracy Round-Off Algorithm for Cyclic Scheduling and Set Covering," Operations Research, vol. 29, no. 3, pp. 501-519, 1981.

37. Bellmore, M., Greenberg, H. J. and Jarvis, J. J., "Multicommodity Disconnecting Sets," Management Science, vol. 16, no. 6, pp. B427-B433, 1970.

38. Aneja, Y. P. and Vemuganti, R. R., "A Row Generation Scheme for Finding a Multi-Commodity Minimum Disconnecting Set," Management Science, vol. 23, no. 6, pp. 652-659, 1977.

39. Valenta, J. R., Capital Equipment Decisions: A Model for Optimal Systems Interfacing, M.S. Thesis, Massachusetts Institute of Technology, June 1969.

40. Revelle, C., Marks, D. and Liebman, J. C., "An Analysis of Private and Public Sector Location Models," Management Science, vol. 16, no. 12, pp. 692-707, 1970.

41. Garfinkel, R. S., "Branch and Bound Methods for Integer Programming," Combinatorial Optimization, John Wiley and Sons, 1979.

42. Brown, G. and Graves, G., "Design and Implementation of a Large Scale (Mixed Integer, Nonlinear) Optimization System," presented at Las Vegas ORSA/TIMS Conference, November 1975.

43. Garfinkel, R. S. and Nemhauser, G. L., Optimal Set Covering: A Survey, Perspectives on Optimization, Addison-Wesley, pp. 164-183, 1972.

44. Kovac, B. L., A New Solution for the General Set Covering Problem, Computer and Automation Institute, Hungarian Academy of Sciences, 1973.

45. Hey, A. M., Algorithm for the Set Covering Problem, PH.D. Dissertation, University of London, 1981.

46. Naval Postgraduate School Report NPS55-84-020, Extracting Embedded Generalized Networks from Linear Programming Problems, by G. Brown , R. McBride and K. Wood, September 1984.

47. Marsten, M., Graph Coloring and Related Problems in OR, Ph.D. Dissertation, Department of Management Science, Imperial College, London, 1975.

48. Balas, E., "Cutting Planes from Conditional Bounds: A New Approach to Set Covering," Mathematical Programming, vol. 12, pp. 19-36, 1980.

49. Fulkerson, D. R., Nemhauser, G. L. and Trotter, L. E., "Two Computationally Difficult Set Covering Problems That Arise in Computing the 1-Width of Incidence Matrices of Steiner Triple Systems," Mathematical Programming Study, vol. 2, pp. 72-81, 1974.

50. Marsten, R. E., An Implicit Enumeration Algorithm for the Set Partitioning Problem with Side Constraints, Ph.D. Dissertation, University of California, Los Angeles, 1971.

51. Vasko, F. J. and Wilson, G. R., "Using a Facility Location Algorithm to Solve Large Set Covering Problems," Operations Research Letters, vol. 3, no. 2, pp. 85-90, June 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
3.	Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, CA 93943	1
4.	Professor Kevin Wood Code 55wd Department of Operations Research Naval Postgraduate School Monterey, CA 93943	10
5.	Professor Gerald G. Brown Code 55bw Department of Operations Research Naval Postgraduate School Monterey, CA 93943	1
6.	Professor Garret N. Vanderplaats Department of Mechanical Engineering University of California, Santa Barbara Santa Barbara, CA 93106	1
7.	Major Kook Jin Nam 19Tong 8Ban 1058-11 Pyeong-Lee 1 Dong, Seo-Gu, Taegu 630, Seoul Korea.	9
8.	Library P.O. BOX 77 GongNeung Dong, Dobong-Gu Seoul 130-09, Korea	2
9.	Major Tae Nam Ahn P.O. BOX 77 Computer Center GongNeung Dong, Dobong-Gu Seoul 130-09, Korea	1
10.	Major Hee Sun Song SMC 1359 Naval Postgraduate School Monterey, CA 93943	1

1 3 5 3 7 5

210473

Thesis
N2393 Nam
c.1
Solving set covering problems using heuristics with branch and bound.

210473

Thesis
N2393 Nam
c.1
Solving set covering problems using heuristics with branch and bound.

thesN2393
Solving set covering problems using heur
3 2768 001 00825 3
DUDLEY KNOX LIBRARY