AD-A102 583 GEORGE WASHINGTON UNIV WASHINGTON DC PROGRAM IN LOGISTICS F/G 12/1
SOLVING MULTIACTIVITY MULTIFACILITY CAPACITY-CONSTRAINED 0-1 AS--ETC(U)
MAY 81 K L CHHABRA N00014-80-C-0169
UNCLASSIFIED SERIAL-T-441 NL

1 OF 2
AD A
102583

THE GEORGE WASHINGTON UNIVERSITY

STUDENTS FACULTY STUDY R
ESEARCH DEVELOPMENT FUT
URE CARER CREATIVITY CO
MMUNITY LEADERSHIP TECH
NOLOGY FRONTI ESIGN
ENGINEERING APP ENG
GEORGE WASHI NIV

SCHOOL OF ENGINEERING AND APPLIED SCIENCE

(12)

SOLVING MULTIACTIVITY MULTIFACILITY CAPACITY-CONSTRAINED 0-1 ASSIGNMENT PROBLEMS

by

Krishan Lal Chhabra

Serial T-441
12 May 1981

The George Washington University
School of Engineering and Applied Science
Institute for Management Science and Engineering

Program in Logistics
Contract N00014-80-C-0169
Project NR 347 059
Office of Naval Research

NONE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER T-441	2. GOVT ACCESSION NO. AD-A102 583	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOLVING MULTIACTIVITY MULTIFACILITY CAPACITY-CONSTRAINED 0-1 ASSIGNMENT PROBLEMS		5. TYPE OF REPORT & PERIOD COVERED SCIENTIFIC
		6. PERFORMING ORG. REPORT NUMBER T-441
7. AUTHOR(s) KRISHAN LAL CHHABRA		8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0169
9. PERFORMING ORGANIZATION NAME AND ADDRESS THE GEORGE WASHINGTON UNIVERSITY PROGRAM IN LOGISTICS✓ WASHINGTON, DC 20052		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS OFFICE OF NAVAL RESEARCH CODE 434 ARLINGTON, VA 22217		12. REPORT DATE 12 May 1981
		13. NUMBER OF PAGES 121
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) NONE
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC SALE AND RELEASE; DISTRIBUTION IS 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)

BRANCH AND BOUND
INTEGER PROGRAMMING
MULTIACTIVITY MULTIFACILITY ASSIGNMENT PROBLEMS
0-1 ASSIGNMENT PROBLEMS

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

A branch-and-bound solution algorithm and a computer program implementing this algorithm are developed to solve multiactivity multifacility capacity-constrained 0-1 assignment problems. Such 0-1 integer programming problems have the objective of minimizing the sum of variable costs due to the assignment of the activities to designs and fixed costs due to the inclusion of the facilities chosen. The constraints ensure

Continued

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

NONE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract - (Cont'd)

that each activity is assigned to a single design and that the capacities of the facilities chosen are not exceeded. Each design involves the use of one or more facilities, and the same design may be used by several activities. This document includes formulation of the problem, mathematical development of the branch-and-bound solution algorithm, a detailed test example, and computational test results using the computer program. The areas of application are identified, and consideration for further improvement of the branch-and-bound solution algorithm are also included.

THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Institute for Management Science and Engineering

Program in Logistics

Abstract
of
Serial T-441
12 May 1981

SOLVING MULTIACTIVITY MULTIFACILITY CAPACITY-CONSTRAINED 0-1 ASSIGNMENT PROBLEMS

by

Krishan Lal Chhabra

A branch-and-bound solution algorithm and a computer program implementing this algorithm are developed to solve multiactivity multifacility capacity-constrained 0-1 assignment problems. Such 0-1 integer programming problems have the objective of minimizing the sum of variable costs due to the assignment of the activities to designs and fixed costs due to the inclusion of the facilities chosen. The constraints ensure that each activity is assigned to a single design and that the capacities of the facilities chosen are not exceeded. Each design involves the use of one or more facilities, and the same design may be used by several activities. This document includes formulation of the problem, mathematical development of the branch-and-bound solution algorithm, a detailed test example, and computational test results using the computer program. The areas of application are identified, and consideration for further improvement of the branch-and-bound solution algorithm are also included.

Program in Logistics

Contract N00014-80-C-0169
Project NR 347 059
Office of Naval Research

SOLVING A MULTIACTIVITY MULTIFACILITY CAPACITY-CONSTRAINED 0-1 ASSIGNMENT PROBLEM

by

Krishan Lal Chhabra
B.M.E. 1965, University of Delhi
M.S. 1973, The George Washington University

A Dissertation submitted to

The Faculty of

The School of Engineering and Applied Science

of The George Washington University in partial satisfaction

of the requirements for the degree of Doctor of Science

May 3, 1981

Dissertation directed by

Richard Martin Soland

Professor of Operations Research

Abstract

SOLVING A MULTIACTIVITY MULTIFACILITY
CAPACITY-CONSTRAINED 0-1 ASSIGNMENT PROBLEM

by

Krishan Lal Chhabra

Richard Martin Soland, Director of Research

A branch-and-bound solution algorithm and a computer program implementing this algorithm are developed to solve a multiactivity multifacility capacity constrained 0-1 assignment problem. The mathematical formulation for such a problem, called problem (P), is to find x_{ij} and y_k values that:

$$\text{Minimize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} + \sum_{k=1}^{p} b_k y_k \qquad \text{(i)}$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_{ij} = 1 \qquad j=1,\dots,n \qquad \text{(ii)}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} d_{ijk} x_{ij} \le s_k y_k \qquad k=1,\dots,p \qquad \text{(iii)}$$

$$x_{ij} = 0 \text{ or } 1 \text{ for all } i \text{ and } j \qquad \text{(iv)}$$

$$y_k = 0 \text{ or } 1 \text{ for all } k \qquad \text{(v)}$$

where i, j, k are indices for designs, activities, and facilities, respectively; x_{ij} has value 1 if and only if activity j uses design i, and y_k has value 1 if and only if facility k is used. A design involves the use of one or more facilities, and the same design may be used by several activities.

Problem (P) has the objective of minimizing the sum of a_{ij}'s -- the variable costs due to the assignments of activities to designs, and b_k's -- the fixed costs due to the facilities used. Constraints (ii) and (iv) ensure that each activity is assigned to a single design. Each d_{ijk} is the capacity required at facility k if activity j uses design i, and is thus equal to zero if design i does not involve the use of facility k. Constraints (iii), therefore, ensure that for each facility k used, the total capacity required does not exceed the capacity available s_k. The difficulty in solving problem (P) stems from the indirect relationship between the assignments and facilities, i.e., an assignment $x_{ij} = 1$ bears on all the constraints (iii) for which d_{ijk} is positive, and, therefore, on several y_k variables.

The branch-and-bound solution algorithm uses Lagrangian relaxation as a basic step in obtaining lower bounds. In addition, it includes several operational rules, such as a branching rule for a judicious choice of the branching variable, a capacity rule to eliminate infeasible assignments, and a bounding rule to eliminate non-optimal assignments.

This dissertation includes relevant background leading to the formulation of problem (P), mathematical development of the branch-and-bound solution algorithm, a detailed test example, and computational test results using the computer program. The areas of application are identified, and suggestions for further improvement of the branch-and-bound solution algorithm are included.

The computer program has been written in FORTRAN IV. A detailed description of the computer program and guidelines for its use are included in a separate document entitled "Program Description and User's Guide for ZIPCAP--a Zero-one Integer Program to solve multiactivity multifacility Capacity-constrained Assignment Problems." Although developed for capacitated problems, the computer program can also be used to solve uncapacitated problems in which it is assumed that the facilities have infinite capacity.

ACKNOWLEDGMENTS

I wish to express my deep gratitude and appreciation to my research director, Professor Richard M. Soland, for introducing me to this problem, providing numerous insights and careful direction, and being extremely generous in sparing his valuable time throughout the research effort. I am very grateful to my long-time academic adviser, as well as research adviser, Professor Donald Gross, for his invaluable advice and guidance, both academic and personal, throughout my graduate program.

Most of this research effort has been supported by the Office of Naval Research under Contract No. N00014-75-C-0729 for which I am greatly indebted to Mr. Robert K. Lehto and Mr. Charlie McPeters (Department of the Navy), and Professor William H. Marlow.

Professors James E. Falk and Garth P. McCormick were kind enough to review this dissertation, and I am very thankful to them for their helpful comments.

I would like to thank Mr. William Caves for his assistance in the development and testing of the computer program, and Professor Charles Pinkus for providing data for the test problems.

I am very thankful to Bettie Taggart and Teresita Abacan for an excellent job in editing and typing.

I take this opportunity to thank my parents and my brothers for their assistance and guidance in my education. Finally, I owe special thanks to my wife Promila who deserves a great part of the credit for her understanding, patience, and encouragement; and to my children Vinita, Adhuna, and Nipun for "letting daddy do his homework" over a long period of time.

TABLE OF CONTENTS

TABLES

1. INTRODUCTION

Multiactivity multifacility assignment problems arise in such diverse areas as public health care systems and private multi-echelon inventory/distribution systems. Such systems involve the assignment of activities or tasks to groups of facilities in such a way that total system cost is minimized. The total system cost has components (fixed costs) that depend on the facilities actually used as well as components (variable costs) that depend solely on the assignment made. Most recently [Gross, Pinkus, and Soland (1979)] there has been interest in including facility capacity constraints as well. For this kind of problem, i.e., a multiactivity multifacility capacity-constrained 0-1 assignment problem, we have developed a solution algorithm of the branch-and-bound type and a computer program based upon it.

The computer program and guidelines for its use are described in a separate document [Chhabra and Soland (1980)] titled "Program Description and User's Guide for ZIPCAP -- a Zero-one Integer Program to solve multiactivity multifacility Capacity-constrained Assignment Problems."

This document describes the development of the solution algorithm and computational test results using the computer program. Suggestions for further improvement in the solution algorithm are also included.

This chapter reviews the relevant literature, provides background leading to the mathematical formulation of the multiactivity multifacility capacity-constrained 0-1 assignment problem, called problem (P), and includes potential areas of application. The theoretical base for developing the algorithm/methodology are described in Chapter 2. Various components of the methodology are covered in detail in Chapter 3. Chapter 4 provides an overview of the computational procedure and the computer program, whereas computational test results are given in Chapter 5. Suggestions for further research and potential improvements in the algorithm are included in Chapter 6.

It may be noted that the basic terminology, described below, in the formulation of problem (P) includes: activities that must be assigned, facilities which serve the activities, designs involving one or more facilities, fixed costs associated with the facilities, and variable costs associated with the assignment of activities to designs.

The following review of the relevant literature starts with the classical assignment problem and leads to the formulation of problem (P). Different authors have used various terminologies in describing relevant formulations. In the following discussion, the original terminologies are used, and are followed by our equivalent terminology, where appropriate, shown in parenthesis.

1.1 Generalized Assignment Problem

In a classical assignment problem [Hillier and Lieberman (1980)], the purpose is to find optimal pairs of agents and tasks or activities. Each task is assigned to a single agent, and each agent is given a single task, and the suitability of a particular set of assignments is determined by a single criterion function such as minimization of cost. In a generalized assignment problem (GAP), several tasks can be assigned each agent, subject to the resources available to the various agents [Ross and Soland (1975)], e.g., assigning software development tasks to programmers and assigning jobs to computers in a computer network.

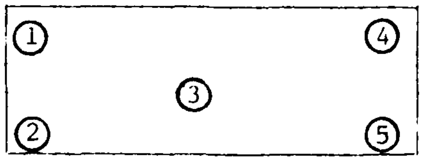
A variety of well-known facility location and location-allocation problems have been shown to be equivalent to, and therefore solvable as GAP's [Ross and Soland (1977)]. Here, in general, the tasks represent demand centers for a good or service, and the agents represent supply centers to be established at potential sites or locations. Each demand center must be supplied from a supply center. A fixed cost is incurred for each supply center established, and, in addition, there is a cost incurred for each unit processed at a supply center and transportation costs incurred for the units sent from supply centers to demand centers. The problem may be "uncapacitated" -- when there is no limit to the number of units that may be processed by a supply center, or "capacitated" -- when there are restrictions on the number of units that may be processed. The objective is to select supply center locations and set up a distribution assignment so that the total cost is minimized.

1.2 Multiactivity Multifacility Uncapacitated Assignment Problem

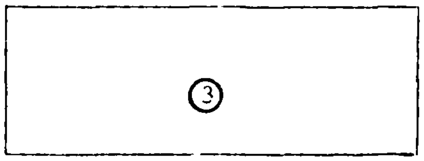
A salient feature of the above facility location problems is that each demand center (activity) is assigned to a single supply center (facility). Sometimes, however, it may be desirable to assign an activity to more than one facility. This leads to the concept of "design," and the multiactivity multifacility assignment problem [Pinkus, Gross, and Soland (1973)]. Before describing such a problem, some terminology is considered first.

A design involves the use of one or more facilities, and represents a meaningful configuration of facilities along with a meaningful strategy for using them -- as illustrated in the following examples.

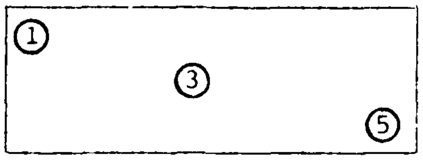
Consider five facilities and their locations as shown in Figure 1(a). (From practical considerations, these may be existing and/or potential locations.) Three of the possible designs are shown in Figures 1(b) to 1(d). Design 1 is completely centralized since it uses only one facility, whereas design 3 is completely decentralized since it uses all the facilities.



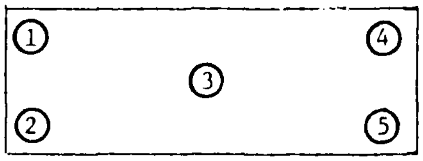
(a) Locations for five facilities



(b) Design 1: one facility--location ③



(a) Design 2: three facilities--locations ①, ③, ⑤



(d) Design 3: five facilities--all locations

Figure 1. Examples of alternate designs for a system of five facilities

It is possible for several designs to have the same facilities but different configuration and strategies for using these facilities, e.g., a multiproduct multi-echelon inventory system [Gross, Pinkus, and Soland (1979)]. Figure 2(a) shows design 1 containing certain facilities (warehouses) at the central, regional, and local levels or echelons. Figure 2(b) shows design 2 with the same facilities but having a different configuration.

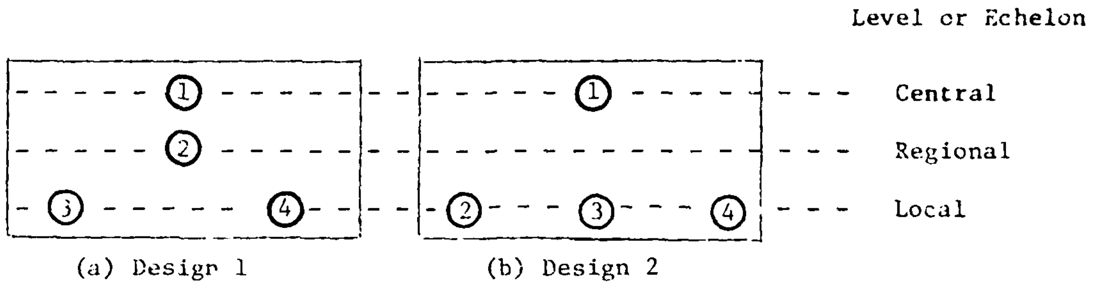


Figure 2. Example of alternative designs having the same facilities but different configuration

The distribution of a given activity at various facilities under design 1 would be different than under design 2, depending, of course, on the inventory policies. This results in different variable costs (described later) for that activity under design 1 as against design 2. In fact, it is possible to have a situation where two or more designs have the same facilities and the same configuration but different strategies, resulting in different variable costs. For example, one strategy might specify an equal distribution of a specific activity over the various facilities, whereas another strategy could impose a different distribution scheme over the same facilities.

In general, if a system is to be composed of at most p facilities, the number of alternative designs is 2^p-1 if no two designs have the same facilities. However, with the same facilities but different configurations and strategies, the number of alternative designs could be much higher. In practice, it is possible to eliminate a majority of alternative designs because of geographical, political, economical, and other factors.

The multiactivity multifacility assignment problem seeks minimization of some measure of total system cost such as, total expected cost over a given time period or total discounted cost over the lifetime of the system. The system cost will include investment costs for building or leasing the system, operating costs for operation and maintenance of the system, and the costs for providing necessary services. Both the investment costs and the operating costs have fixed as well as variable components [Ross and Soland (1980)]. The fixed components include those costs associated with the facilities of a given design which are independent of the activities served. Such costs are called fixed costs. On the other hand, the variable components and the service costs include those costs which are completely dependent on the service demand of the activities at the various facilities in a given design. Such costs are called variable costs. By definition, both the fixed costs and the variable costs are relative terms.

An equivalent formulation of the multiactivity multifacility assignment problem defined by Pinkus, Gross, and Soland (1973) is as follows.

Let a_{ij} = variable cost of activity j using design i $(i=1,\ldots,m;\ j=1,\ldots,n)$

b_k = fixed cost of facility k $(k=1,\ldots,p)$

b_{ik} = 1 if facility k is included in design i ,

= 0 otherwise.

The decision variable x_{ij} is defined as:

$$x_{ij} = 1 \text{ if activity } j \text{ uses design } i \text{ ,}$$
$$= 0 \text{ otherwise.}$$

Then, the uncapacitated assignment problem called problem (PU) is to find x_{ij} values that:

$$(\text{PU})\begin{cases} \text{Minimize} \quad \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} + \sum_{k=1}^{p} b_k u\left(\sum_{i=1}^{m} b_{ik} \sum_{j=1}^{n} x_{ij}\right) & (1) \\ \text{subject to} \quad \sum_{i=1}^{m} x_{ij} = 1 \quad \text{for } j=1,\ldots,n & (2) \\ \quad x_{ij} = 0 \text{ or } 1 \text{ for all } i \text{ and } j & (3) \\ \text{where} \quad u(\cdot) = 0 \text{ if } (\cdot) \leq 0 \text{ ,} & \\ \quad = 1 \text{ if } (\cdot) > 0 \text{ .} & \end{cases}$$

The objective function of this problem consists of two distinct parts. The first part represents the total variable cost, and the second, the total fixed cost of the system. Constraints (2) and (3) ensure that each activity is assigned to a single design. Of course, the optimal solution may involve the use of more than one design.

Problem (PU) is a 0-1 nonlinear programming problem (because of the step function u), and a branch-and-bound algorithm using linear underestimates for the nonlinear part of the objective function has been described in Pinkus, Gross, and Soland (1973). A heuristic procedure for this problem is given by Khumawala and Stinson (1980) in an unpublished paper. This procedure is an extension of some earlier work [Khumawala (1973)].

1.3 Adding Capacity Constraints -- Problem (P)

A weakness of problem (PU) is that it assumes unlimited capacity available at each facility in terms of the activities using a given facility. In practice, a facility may not have the capability to serve every activity, and may have restrictions as to the total capacity available to handle more than one activity.

Let s_k = capacity available at facility k, and

d_{ijk} = capacity required at facility k for activity j when activity j uses design i.

If design i does not include facility k, then $d_{ijk} = 0$ for all j.

Define the decision variable y_k as:

$$y_k = 1 \text{ if facility } k \text{ is used,}$$
$$= 0 \text{ otherwise .}$$

Then the assignment problem [Gross, Pinkus, and Soland (1979)], called problem (P) is to find x_{ij} and y_k values that:

$$(P)\left\{\begin{array}{lll}
\text{Minimize} & \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} + \sum_{k=1}^{p} b_k y_k & (4)\\
\text{subject to} & \sum_{i=1}^{m} x_{ij} = 1 \qquad j=1,\ldots,n & (2)\\
& \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ijk} x_{ij} \le s_k y_k \qquad k=1,\ldots,p & (5)\\
& x_{ij},\ y_k = 0 \text{ or } 1 \text{ for all } i,j,k & (6)
\end{array}\right.$$

Constraints (5) of problem (P) ensure that the capacities available at the facilities are not violated. Problem (P) is, thus, a multiactivity multifacility capacity-constrained 0-1 assignment problem, as compared to problem (PU) which is uncapacitated. In problem (P), constraints (2) along with the part of constraints (6) involving the x_{ij}'s ensure that each activity is assigned to a single design. Of course, the optimal solution may result in the use of more than one design.

For an example of five facilities and three designs as shown in Figures 1(b) to 1(d), and four activities; the matrix $[a_{ij}|b_k|d_{ijk}]$ is as shown in Figure 3.

1.3.1 Comparison with the uncapacitated assignment problem.

Comparison of the capacitated problem (P) with the uncapacitated problem (PU) shows that the objective functions (1) and (4) are equivalent and constraints (2) in each are the same. Constraints (5) serve to impose the capacity constraints and at the same time, for a given design, the relevant facilities are forced in the solution. For an

$$\begin{array}{cc|cccc|ccccc|cccc}
 & & \multicolumn{4}{c|}{\underline{\text{Variable Costs } (a_{ij})}} & \multicolumn{5}{c|}{\underline{\text{Fixed Costs } (b_k)}} & \multicolumn{4}{c}{\underline{\text{Capacities Required } (d_{ijk})}} \\
 & & & & & & & & & & & \multicolumn{4}{c}{\text{e.g., for } k=1^{*}} \\
 & & \multicolumn{4}{c|}{\text{Activities } (j)} & \multicolumn{5}{c|}{\text{Facilities } (k)} & \multicolumn{4}{c}{\text{Activities } (j)} \\
 & & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 5 & 1 & 2 & 3 & 4 \\
 & 1 & a_{11} & a_{12} & a_{13} & a_{14} & & & b_3 & & & 0 & 0 & 0 & 0 \\
\text{Designs } (i) & 2 & a_{21} & a_{22} & a_{23} & a_{24} & b_1 & & b_3 & & b_5 & d_{211} & d_{221} & d_{231} & d_{241} \\
 & 3 & a_{31} & a_{32} & a_{33} & a_{34} & b_1 & b_2 & b_3 & b_4 & b_5 & d_{311} & d_{321} & d_{331} & d_{341}
\end{array}$$

Figure 3.
Matrix of variable costs, fixed costs, and capacities required -- example

**Similar* d_{ijk} *values exist for* $k=2,\dots,5$ depending on the inclusion of the facility in a design.

x_{ij} equal to 1, all the facilities with $d_{ijk} > 0$ must have y_k values equal to 1 in order to satisfy (5) and the corresponding fixed costs b_k are therefore included in (4). If $y_k = 0$ and $d_{ijk} > 0$, then x_{ij} must be 0 in order to satisfy (5).

Problem (P) has been formulated as a 0-1 linear programming problem whereas problem (PU) was formulated as a 0-1 nonlinear programming problem.

Note that problem (PU) can be easily obtained as a special case of problem (P) by letting d_{ijk} equal 1 (for all j) if design i uses facility k, and setting all s_k equal to n. In other words, the corresponding formulation is to find x_{ij} and y_k values that:

$$(\text{PI})\left\{\begin{array}{llll}
\text{Minimize} & \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_{ij} + \sum_{k=1}^{p} b_k y_k & & (4) \\
\text{subject to} & \sum_{i=1}^{m} x_{ij} = 1 & j=1,\ldots,n & (2) \\
 & \sum_{i=1}^{m} e_{ik} \sum_{j=1}^{n} x_{ij} \leq n y_k & k=1,\ldots,p & (7) \\
 & x_{ij}, y_k = 0 \text{ or } 1 \text{ for all } i,j,k & & (6) \\
\text{where} & e_{ik} = 1 \text{ if design } i \text{ uses facility } k, & & \\
 & \quad = 0 \text{ otherwise .} & &
\end{array}\right.$$

1.3.2 Comparison with the fixed-charge location-allocation problem.

Problem (P) bears a resemblance to the well-known fixed-charge location-allocation problem or capacitated facility location problem [Geoffrion (1975); Ross and Soland (1977)]. There are, however, very significant differences between the two. In order to point out these differences, here is a statement of the location-allocation problem (LA) as given by Gross, Pinkus, and Soland (1979) in a form similar to that of problem (P).

Find x_{kj} and y_k values that

$$
(\mathrm{LA}) \left\{ \begin{array}{lll}
\text{Minimize} & \sum_{k=1}^{p} \sum_{j=1}^{n} a_{kj} x_{kj} + \sum_{k=1}^{p} b_k y_k & (8) \\
\text{subject to} & \sum_{k=1}^{m} x_{kj} = 1 \qquad j=1,\ldots,n, & (9) \\
 & \sum_{j=1}^{n} d_j x_{kj} \leq s_k y_k \qquad k=1,\ldots,p & (10) \\
 & x_{kj} \geq 0 , \; y_k = 0 \text{ or } 1 \text{ for all } j \text{ and } k & (11)
\end{array} \right.
$$

Here x_{kj} represents the fraction of customer (activity) j's demand that is supplied by a facility at location k.

The most important distinction between problem (LA) and problem (P) is the relationship between assignments and facilities. In problem (LA) there is a direct connection between the assignments made and the facilities required, and each assignment affects only one facility, i.e., the assignment $x_{kj} > 0$ has a bearing on only one of the constraints (10) and, therefore, on only one variable y_k. On the other hand, in problem (P), the connection between the assignments made and the facilities required is indirect, and each assignment can affect several facilities, i e., the assignment $x_{ij} = 1$ bears on all of the constraints (5) for which $d_{ijk} > 0$ and, therefore, on several variables y_k.

Another distinction is the relative difficulty of the two problems. While problem (LA) is not easy to solve, branch-and-bound approaches have been successful in dealing with it because once values are specified for the y_k, the x_{jk} are found by solving a transportation problem. Problem (LA) becomes more difficult if the constraints $x_{kj} \geq 0$ in (10) are replaced by $x_{kj} = 0$ or 1 in order to preclude supply of customer (activity) j's demand by more than one facility. With this change, problem (LA) may be treated as a generalized assignment problem and is solvable using an efficient branch-and-bound algorithm [Ross and Soland (1977)]. Problem (P) is more difficult than this variation of problem (LA) because of the above stated indirect connection between the assignments and the facilities. Even after values have been specified for all the y_k , problem (P) remains a difficult 0-1 linear programming problem because of the interaction of the constraints.

1.3.3 Solving problem (P).

The capacitated problem (P) has $mn+p$ 0-1 variables and $n+p$ constraints, so the problem dimensions may be large from practical considerations. For example, with $m=n=30$ and $p=20$, problem (P) has 920 variables and 50 constraints. The 0-1 LP computer codes generally available are limited in terms of problem size. For example, the code used by Gross, Pinkus, and Soland (1979) can handle up to 40 variables and 20 constraints. A better and more efficient code [Geoffrion and Nelson (1968)] allows up to 90 variables and 50 constraints. This fact, together with the structure of problem (P) suggests that a specialized algorithm could be developed that would be more efficient for practical problems than the general integer linear programming algorithms (on which the available codes are based).

With the above background in mind, the development of the solution algorithm and the computer program to solve problem (P) was undertaken and is described in Chapters 2 through 4.

1.4 Areas of Application

The solution algorithm and the computer program are designed to solve a multiactivity multifacility capacity-constrained 0-1 assignment problem, i.e., one which can be formulated as problem (P).

The basic elements of such a problem are activities that must be assigned, facilities and their meaningful configurations represented as designs, the fixed and variable costs, and the capacity requirements of the activities.

The formulation (P) applies to existing and/or proposed facilities. In other words, it is useful for a situation where the decision may be to delete some of the existing facilities, as well as for a situation where the decision involves a selection out of a set of proposed facilities.

Table 1 includes examples of areas where formulation (P) is applicable. Within each application area, activities and facilities are identified. The implications of designs, variable costs, and fixed costs are apparent.

Obtaining the values of the data elements b_k, d_{ijk}, s_k, and in particular a_{ij}, can be a simple or a complex exercise depending on the particulars of the application, and the nature of the components comprising these elements. For example, in designing multi-echelon inventory systems [Gross, Pinkus, and Soland (1979)], a_{ij} represents the inventory cost of product (activity) j using echelon structure (design) i and b_k represents the fixed cost of installation (facility) k. The inventory cost a_{ij} includes the cost of procurement, carrying inventory, filling orders, and stockouts. The value a_{ij}, and associated inventory stockage policies, are arrived at by solving a multi-echelon inventory problem. In other words, for product j stocked under echelon structure i, optimal inventory policies are determined, at each installation of the structure, which yields a_{ij}. The facility fixed cost b_k includes the capital expenditure for building the installation, along with a number of fixed costs associated with operating it, such as administrative expenses, the expense of renting the facility (if it is not built), and certain other fixed operating expenses.

In the case of designing a support system for repairable

TABLE 1

EXAMPLES OF APPLICATION AREAS

Area of Application	Activities	Facilities
1. Design of multi-echelon inventory systems*	Types of items to be stocked	Warehouses (Comprising different levels or echelons, e.g., central, regional, and local warehouses)
2. Assignment of repairable components**	Major components of a unit, e.g., components of an aircraft, a ship, a piece of machinery	Repair depots
3. Design of training programs	Training program categories or occupational classifications	Training schools
4. Location of facilities***	Types of services, e.g., health-care services	Buildings or installations, e.g., health-care centers

*Gross, Pinkus, and Soland (1979)

**Gross and Pinkus (1979)

***Pinkus, Gross, and Soland (1973)

items [Gross and Pinkus (1979)], a_{ij} represents the total variable cost if unit type (activity) j is repaired under design i . The set of parameters taken into consideration to compute this cost for each unit type includes such things as varying population sizes, failure rates, average repair times, costs associated with their repair, the purchase and storage of spares, the purchase of repair channels, and travel to depots (facilities) for repair. A computer program is used to solve a spares and server provisioning problem, and the results provide the basic information to compute a_{ij} .

Thus, in general, the data elements of problem (P) may be obtained directly and/or by solving other related problem(s); it depends on the definition and the nature of the components comprising these data elements for a specific application area.

2. DEVELOPMENT OF THE SOLUTION ALGORITHM

The solution algorithm that has been developed to solve problem (P) is a branch-and-bound procedure which makes use of Lagrangian relaxation as a basic step.

This chapter considers two different Lagrangian relaxations of problem (P), their general characteristics, and some useful results leading to the specific case of Lagrangian relaxation utilized in the solution algorithm.

2.1 Lagrangian Relaxation

Taking a set of "complicating" constraints of a general mixed-integer program into the objective function in a Lagrangian fashion (with fixed multipliers) results in a "Lagrangian relaxation" of the original problem [Geoffrion (1974)]. The relaxed problem is easy to solve compared to the original problem, and provides a lower bound (for minimization problems) on the optimal value of the original problem.

Although the use of Lagrangian relaxation in discrete optimization has been reported prior to 1970 [e.g., Lorie and Savage (1955), Everett (1963), and Gilmore and Gomory (1963)], the "birth" of the Lagrangian approach as it exists today [Fisher (1978)] occurred in 1970 with the successful application of Lagrangian relaxations to the traveling salesman problem [Held and Karp (1970, 1971)]. This was followed by application of Lagrangian relaxation to scheduling problems [Fisher and Schrage (1972), and Fisher (1973, 1976)], the general integer programming problem [Shapiro (1971), and Fisher and Shapiro (1974)] and the generalized assignment problem [Ross and Soland (1975)]. Table 2 lists the applications of Lagrangian relaxation as given by Fisher (1978). A review of Lagrangian relaxation is also provided by Shapiro (1977) and Christofides (1980).

TABLE 2

APPLICATIONS OF LAGRANGIAN RELAXATION*

Problem	Researchers	Lagrangian Problem
TRAVELING SALESMAN		
Symmetric	Held & Karp (1970, 1971)	Spanning Tree
Asymmetric	Bazarra & Goode (1977)	Spanning Tree
Symmetric	Balas & Christofides (1976)	Perfect 2-Matching
Asymmetric	Balas & Christofides (1976)	Assignment
SCHEDULING		
n|m Weighted Tardiness	Fisher (1973)	Pseudo-Polynomial Dynamic Programming
1 Machine Weight Tardiness	Fisher (1976)	Pseudo-Polynomial DP
Power Generation Systems	Muckstadt & Koenig (1977)	Pseudo-Polynomial DP
GENERAL IP		
Unbounded Variables	Fisher & Shapiro (1974)	Group Problem
Unbounded Variables	Burdet & Johnson (1976)	Group Problem
0 - 1 Variables	Etcheberry, et. al. (1978)	0 - 1 GUB
LOCATION		
Uncapacitated	Cornuejols, Fisher, & Nemhauser (1977)	0 - 1 VUB
Capacitated	Geoffrion & McBride (1977)	0 - 1 VUB
Databases in Computer Networks	Fisher & Hochbaum (1978)	0 - 1 VUB
GENERALIZED ASSIGNMENT		
	Ross & Soland (1975)	Knapsack
	Chalmet & Gelders (1976)	Knapsack, 0-1 GUB
SET COVERING--PARTITIONING		
Covering	Etcheberry (1977)	0 - 1 GUB
Partitioning	Nemhauser & Weber (1978)	Matching

*Source: Fisher (1978)

2.1.1 Relaxing Problem (P)

By dividing constraints (5) by s_k and letting $r_{ijk} = d_{ijk}/s_k$, problem (P) can be restated as follows.

$$
(P)\left\{\begin{array}{llll}
\text{Minimize} & \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}\, x_{ij} + \sum_{k=1}^{p} b_k y_k & & (4)\\
\text{subject to} & \sum_{i=1}^{m} x_{ij} = 1 & j=1,\ldots,n & (2)\\
& \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ijk}\, x_{ij} \le y_k & k=1,\ldots,p & (5')\\
& x_{ij},\ y_k = 0 \text{ or } 1 & \text{for all } i,j,k & (6)
\end{array}\right.
$$

A Lagrangian relaxation (LR_u) of problem (P) relative to constraints (2) is obtained as

$$
(LR_u)\left\{\begin{array}{llll}
\text{Minimize} & \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}\, x_{ij} + \sum_{k=1}^{p} b_k y_k - \sum_{j=1}^{n} u_j \left(\sum_{i=1}^{m} x_{ij} - 1\right) & & (12)\\
\text{subject to} & \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ijk}\, x_{ij} \le y_k & k=1,\ldots,p & (5')\\
& x_{ij},\ y_k = 0 \text{ or } 1 & \text{for all } i,j,k & (6)
\end{array}\right.
$$

where the u_j are Lagrange multipliers; it follows that the optimal value of problem (LR_u) is a lower bound on the optimal value of problem (P), i.e., $Z(LR_u) \le Z(P)$. We will continue to use this notation in which $Z(\cdot)$ is the optimal value of problem $(\cdot)$.

Another Lagrangian relaxation (LR_v) of problem (P), relative to constraints (5'), is obtained as

$$\text{Minimize} \quad \sum_i \sum_j a_{ij} x_{ij} + \sum_k b_k y_k - \sum_k v_k \left(y_k - \sum_i \sum_j x_{ij} r_{ijk} \right)$$

subject to (2) and (6), or equivalently,

$$(LR_v) \begin{cases} \text{Minimize} & \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k v_k r_{ijk} \right) - \sum_k y_k \left(v_k - b_k \right) & (13) \\ \text{subject to} & \sum_i x_{ij} = 1 \qquad j=1,\dots,n & (2) \\ & x_{ij},\ y_k = 0 \text{ or } 1 \qquad \text{for all } i,j,k & (6) \end{cases}$$

where the v_k are non-negative Lagrange multipliers; it follows that $Z(LR_v) \leq Z(P)$.

2.1.2 General Characteristics

A Lagrangian relaxation provides a lower bound on the optimal value of the orginal problem, i.e., in our case $Z(LR_u) \leq Z(P)$ and $Z(LR_v) \leq Z(P)$. The usefulness of a Lagrangian relaxation depends on the closeness of this lower bound to the optimal value of the orginal problem. However, the relaxation must be "easy" to solve relative to the original problem. We observe that the optimal value of y_k in problem (LR_v) is 1 if $(v_k - b_k) \geq 0$ and 0 if $(v_k - b_k) \leq 0$, and then problem (LR_v) reduces to n 0-1 "multiple choice" problems which are very easy to solve. On the other hand, problem (LR_u) reduces to k 0-1 knapsack problems. However, these problems are not independent because of the interaction of constraints (5') and the indirect relationship described earlier in Section 1.3 between the assignments and the facilities. In view of this complexity, relaxation (LR_u) will not be considered further.

The choice of Lagrange multipliers in relaxation (LR_v) should be such that $Z(LR_v)$ is as large as possible and hence as close as possible to $Z(P)$ in view of the relationship $Z(LR_v) \leq Z(P)$. In other words, an equivalent problem is to find a vector v (representing v_1, v_2, $\dots, v_k$) to

$$(D)\quad \begin{cases} \text{Maximize} & \left[Z(LR_v)\right] \\ & v \geq 0 \end{cases} \tag{14}$$

Obviously, $Z(LR_v) \leq Z(D) \leq Z(P)$.

The general properties of Lagrangian relaxation have been well described in the literature [e.g., Geoffrion (1974), Geoffrion and McBride (1978), and Fisher (1980)]. Some of these properties relating the Lagrangian relaxation and the usual LP relaxation are stated below.

The LP relaxation $(\bar{P})$ of problem (P) is obtained by relaxing the integrality constraints (6), i.e., the formulation $(\bar{P})$ is

$$(\bar{P})\quad \begin{cases} \text{Minimize} & \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} + \sum_{k=1}^{p} b_k y_k & & (4) \\ \text{subject to} & \sum_{i=1}^{m} x_{ij} = 1 & j=1,\dots,n & (2) \\ & \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ijk} x_{ij} \leq y_k & k=1,\dots,p & (5') \\ & y_k \leq 1 & k=1,\dots,p & (15) \\ & x_{ij}, y_k \geq 0 & \text{for all } i,j,k & (16) \end{cases}$$

Note that the constraints $x_{ij} \leq 1$ are implicit in constraints (2).

Also consider the following partial convex hull relaxation (P^*) of problem (P).

$$(P^*)\quad \begin{cases} \text{Minimize} & \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_{ij} + \sum_{k=1}^{p} b_k y_k & & (4) \\ \text{subject to} & \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ijk} x_{ij} \leq y_k & k=1,\dots,p & (5') \\ & x_{ij}, y_k \in \text{convex hull } \{(2), (6)\} & & (17) \end{cases}$$

Then the relationships between the optimal values of various problems [Geoffrion and McBride (1978)] are as follows.

$$Z(\bar{P}) \leq Z(LR_{\tilde{v}}) \leq \max_{v \geq 0} Z(LR_v) = Z(D) = Z(P^*) \leq Z(P) \qquad (18)$$

where $\tilde{v}$ are the values $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_p$ of a dual optimal solution of $(\bar{P})$ corresponding to constraints (5').

Thus, the optimal dual solution associated with the usual LP relaxation furnishes a choice of Lagrange multipliers such that the associated Lagrangian relaxation is at least as tight as the usual LP relaxation, and generally a good deal tighter and even as tight as the partial convex hull relaxation.

Since $Z(D) = Z(P^*)$, the quality of the bound obtained from the Lagrangian relaxation depends on where $Z(P^*)$ lies in the range between $Z(\bar{P})$ and $Z(P)$. It turns out that problem (LR_v) possesses the "integrality property," i.e., the optimal value of problem (LR_v) is not altered by dropping the integrality conditions on its variables and therefore [Geoffrion (1974)]

$$Z(D) = Z(P^*) = Z(\bar{P}) \qquad (19)$$

Thus, the Lagrangian relaxation (LR_v) is no better than the LP relaxation $(\bar{P})$. On the other hand, Lagrangian relaxation (LR_u) does not possess the integrality property and, hence, could provide an equal or better bound than the LP relaxation $(\bar{P})$; but the computational difficulties do not favor pursuing formulation (LR_u) .

It is possible to consider alternative formulations of problem (P) with the objective of obtaining tighter bounds. This aspect is discussed in Chapter 6.

2.2 Some Results

We now turn to the basic question of choosing Lagrange multipliers v so that (LR_v) is optimal to the extent possible, which is equivalent to solving problem (D). We also need to consider this question when some of the x_{ij} and y_k variables have been assigned values of 1 or 0, i.e., at a node other than the starting or "root" node in the branch-and-bound tree. For this purpose, some terminology is defined and formulations corresponding to problems (P), (LR_v) and (D) are first developed. Then some important results pertaining to the choice of Lagrange multipliers will be proved. Gavish (1978) provides a method of obtaining the 'best' multipliers, based on solving an equivalent linear programming problem. Such a formulation is difficult in our case, and, besides, we propose to avoid solving LP problems in our branch-and-bound procedure.

Define the sets

$S = \{(i,j) | x_{ij}$ has an assigned value of 1 or 0$\}$, and

$T = \{k | y_k$ has an assigned value of 1 or 0$\}$.

These sets represent the partial solution of problem (P) and the variables contained in these sets are termed fixed variables. [Geoffrion (1967)]. Let $\bar{S}$ and $\bar{T}$ represent the corresponding complementary sets, i.e., comprised of the x_{ij} and y_k variables, which have not been assigned specific values and, therefore, are called free variables. A completion of a partial solution is defined as a solution that is determined by S and T together with a binary specification (0 or 1) of the values of the free x_{ij} and y_k variables from sets $\bar{S}$ and $\bar{T}$.

Let $S \cup \bar{S} = S_1$ and $T \cup \bar{T} = T_1$.

Consider a partial solution to problem (P) in which specific values (of 1 or 0) are assigned to some of the x_{ij} and y_k such that

$$\sum_{\substack{i=1\\(i,j)\in S}}^{m} x_{ij} \le 1 \qquad \forall j ,$$

and
$$\sum_{\substack{i\ j\\(i,j)\in S}}\sum r_{ijk}\, x_{ij} \le y_k \qquad \forall k\in T$$

$$\sum_{\substack{i\ j\\(i,j)\in \bar{S}}}\sum r_{ijk}\, x_{ij} \le 1 \qquad \forall k\in \bar{T}$$

and such that $x_{ij} = 1$ and $e_{ik} = 1$ imply that $k\in T$ and $y_k = 1$. Recall that, by definition, $e_{ik} = 1$ if design i uses facility k, and $e_{ik} = 0$ otherwise.

The problem of finding an optimal completion of the partial solution of problem (P) can be stated as follows.

$$(P_\ell)\left\{\begin{array}{lll}
\text{Minimize} & \displaystyle\sum_{\substack{i\ j\\(i,j)\in \bar{S}}}\sum a_{ij}\, x_{ij} + \sum_{\substack{k\\k\in \bar{T}}} b_k y_k + \sum_{\substack{i\ j\\(i,j)\in S}}\sum a_{ij}\, x_{ij} + \sum_{\substack{k\\k\in T}} b_k y_k & (20)\\[2ex]
\text{subject to} & \displaystyle\sum_{\substack{i\\(i,j)\in \bar{S}}} x_{ij} = 1 - \sum_{\substack{i\\(i,j)\in S}} x_{ij} \qquad \forall j & (21)\\[2ex]
 & \displaystyle\sum_{\substack{i\ j\\(i,j)\in \bar{S}}}\sum r_{ijk}\, x_{ij} \le y_k - \sum_{\substack{i\ j\\(i,j)\in S}}\sum r_{ijk}\, x_{ij} \qquad \forall k & (22)\\[2ex]
 & x_{ij},\ y_k = 0 \text{ or } 1 \qquad \forall (i,j)\in \bar{S},\ k\in \bar{T} & (23)
\end{array}\right.$$

We call this problem (P_ℓ) where ℓ indicates the node in the branch-and-bound tree.

A Lagrangian relaxation of problem (P_ℓ) with respect to constraints (22) is obtained by introducing non-negative Lagrange multipliers v_k, $k=1,2,\ldots,p$; the relaxation is then

$$\text{Minimize} \quad \sum_{\substack{i\ j \\ (i,j)\in\bar{S}}} a_{ij}\, x_{ij} + \sum_{\substack{k \\ k\in\bar{T}}} b_k y_k + \sum_{\substack{i\ j \\ (i,j)\in S}} a_{ij}\, x_{ij} + \sum_{\substack{k \\ k\in T}} b_k y_k$$

$$- \sum_k v_k \left[y_k - \sum_{\substack{i\ j \\ (i,j)\in\bar{S}}} r_{ijk}\, x_{ij} - \sum_{\substack{i\ j \\ (i,j)\in S}} r_{ijk}\, x_{ij} \right] \qquad (24)$$

$$\text{subject to} \quad \sum_{\substack{i \\ (i,j)\in\bar{S}}} x_{ij} = 1 - \sum_{\substack{i \\ (i,j)\in S}} x_{ij} \qquad \forall j \qquad (21)$$

$$x_{ij},\ y_k = 0 \text{ or } 1 \qquad \forall (i,j)\in\bar{S}\ ,\ k\in\bar{T} \qquad (23)$$

Rearranging (24), and using the relationship $T_1 = T \cup \bar{T}$, we have problem $(LR_{\ell,v})$:

$$(LR_{\ell,v}) \begin{cases} \text{Minimize} \quad \displaystyle\sum_{\substack{i\ j \\ (i,j)\in\bar{S}}} x_{ij} \left(a_{ij} + \sum_{k\in T_1} v_k\, r_{ijk} \right) + \sum_{\substack{i\ j \\ (i,j)\in S}} x_{ij} \left(a_{ij} + \sum_{k\in T_1} v_k\, r_{ijk} \right) \\ \qquad - \displaystyle\sum_{k\in\bar{T}} y_k \left(v_k - b_k \right) - \sum_{k\in T} y_k \left(v_k - b_k \right) & (25) \\ \text{subject to} \quad \displaystyle\sum_{\substack{i \\ (i,j)\in\bar{S}}} x_{ij} = 1 - \sum_{\substack{i \\ (i,j)\in S}} x_{ij} \qquad \forall j & (21) \\ \qquad x_{ij},\ y_k = 0 \text{ or } 1 \qquad \forall (i,j)\in\bar{S}\ ,\ k\in\bar{T} & (23) \end{cases}$$

Then we have $Z(LR_{\ell,v}) \leq Z(P_\ell)$. An important problem is the choice of Lagrange multipliers $v_1, v_2, \ldots, v_p$, represented by vector v , that maximize $Z(LR_{\ell,v})$, i.e., the problem (D_ℓ) :

$$(D_\ell) \begin{cases} \text{Maximize} \left[Z(LR_{\ell,v}) \right] & (26) \\ \quad v \geq 0 \end{cases}$$

We now state and prove some theorems related to the choice of Lagrange multipliers $v_1, v_2, \ldots, v_p$.

<u>Theorem 1</u>: There exists an optimal solution to problem (D) in which $v_k \geq b_k$ for all k .

<u>Proof</u>: Suppose $v_1 < b_1$, in an optimal solution to problem (D), i.e., $Z(D) = Z(LR_{v^*})$ where $v_1^* < b_1$.

Recall that

$$Z(LR_{v^*}) = \text{Min} \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k v_k^* r_{ijk}\right) - \sum_k y_k \left(v_k^* - b_k\right)$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad \forall j \qquad (2)$$

$$x_{ij}, y_k = 0 \text{ or } 1 \qquad \forall i,j,k \qquad (6)$$

For $v_1^* < b_1$, the optimal value of y_1 is 0 , and the term $- y_1 (v_1^* - b_1)$ in the objective function is 0.

Consider what happens if we increase v_1^* to b_1 . Call the resulting vector $\underline{v}$. Consider problem $(LR_{\underline{v}})$. The optimal value of y_1 in problem $(LR_{\underline{v}})$ is 0 or 1 , and the term $- y_1 (\underline{v}_1 - b_1)$ is 0 . However, the optimal value of y_k is the same in problems (LR_{v^*}) and $(LR_{\underline{v}})$ for all $k > 1$. Therefore, the quantity $\sum_k y_k (v_k - b_k)$ is the same at the optimal solution for both $v = v^*$ and $v = \underline{v}$.

Since $\underline{v}_1 > v_1^*$, we note that in the objective function,

$$a_{ij} + \sum_{k=1}^{p} \underline{v}_k r_{ijk} \geq a_{ij} + \sum_{k=1}^{p} v_k^* r_{ijk} \qquad \forall i,j ,$$

and therefore $Z(LR_{\underline{v}}) \geq Z(LR_{v^*})$.

It follows that there is an optimal solution to problem (D) in which $v_1 \geq b_1$.

Since the choice of $k=1$ was arbitrary, the same result holds for any value of k, $k=1,\ldots,p$; hence, there exists an optimal solution to problem (D) in which $v_k \geq b_k$ for all k .

<u>Theorem 2</u>: There exists an optimal solution to problem (D_ℓ) in which $v_k \geq b_k$ if (i) $k \varepsilon \bar{T}$ or (ii) $k \varepsilon T$ and $y_k = 0$.

<u>Proof</u>: Suppose $v_1 < b_1$ in an optimal solution to problem (D_ℓ) , i.e., $Z(D_\ell) = Z(LR_{\ell,v^*})$ where $v_1^* < b_1$. Then $k=1$ can be such that $k \varepsilon T$ or $k \varepsilon \bar{T}$.

<u>Case (i)</u>: Let $k \varepsilon \bar{T}$.

Recall that

$$Z(LR_{\ell,v^*}) = \text{Min} \sum_{\substack{i\ j \\ (i,j)\varepsilon\bar{S}}} \sum x_{ij} \left(a_{ij} + \sum_k v_k^* r_{ijk} \right)$$

$$+ \sum_{\substack{i\ j \\ (i,j)\varepsilon S}} \sum x_{ij} \left(a_{ij} + \sum_k v_k^* r_{ijk} \right)$$

$$- \sum_{k\varepsilon\bar{T}} y_k \left(v_k^* - b_k \right) - \sum_{k\varepsilon T} y_k \left(v_k^* - b_k \right)$$

$$\text{s.t.} \quad \sum_{\substack{i \\ (i,j)\varepsilon\bar{S}}} x_{ij} = 1 - \sum_{\substack{i \\ (i,j)\varepsilon S}} x_{ij} \qquad \forall j \qquad (21)$$

$$x_{ij},\ y_k = 0 \text{ or } 1 \qquad \forall (i,j)\varepsilon\bar{S}\ ,\ k\varepsilon\bar{T} \qquad (23)$$

For $v_1^* < b_1$, and $k \varepsilon \bar{T}$, the optimal value of y_1 is 0 and the term $- y_1 (v_1^* - b_1)$ in the objective function is 0 .

Let v_1^* be increased to b_1 ; call the resulting vector $\underline{v}$. Consider problem $(LR_{\ell,\underline{v}})$. The optimal value of y_1 in $(LR_{\ell,\underline{v}})$ is 0 or 1 , then the term $- y_1 (\underline{v}_1 - b_1)$ is 0 . For $k > 1$, the optimal value of y_k being the same in (LR_{ℓ,v^*}) and $(LR_{\ell,\underline{v}})$, we find that $\sum_{k\epsilon T_1} y_k (v_k - b_k)$ is the same at the optimal solution for both $v = v^*$ and $v = \underline{v}$. But $\underline{v}_1 > v_1^*$; therefore

$$a_{ij} + \sum_{k\epsilon T_1} \underline{v}_k r_{ijk} \geq a_{ij} + \sum_{k\epsilon T_1} v_k^* r_{ijk} \quad \forall(i,j)\epsilon S \text{ and } (i,j)\epsilon \bar{S}$$

Hence, $Z(LR_{\ell,\underline{v}}) \geq Z(LR_{\ell,v^*})$, wherefrom it follows that there is an optimal solution to (D_ℓ) in which $v_1 \geq b_1$. Since the choice of $k=1$ was arbitrary, the same results hold for any value of k, $k\epsilon\bar{T}$. Hence, there exists an optimal solution to problem (D_ℓ) in which $v_k \geq b_k$ for all $k\epsilon\bar{T}$.

Case (ii): Let $k\epsilon T$ and $y_k = 0$

Considering problem (LR_{ℓ,v^*}) , for $k = 1$, $v_1^* < b_1$, and $y_1 = 0$, the term $- y_1(v_1^* - b_1)$ in the objective function is 0 .

Increase v_1^* to b_1 and call the resulting vector $\underline{v}$. The term $- y_1(\underline{v}_1 - b_1)$ is 0 . For $k > 1$, the optimal values of y_k are the same in problems (LR_{ℓ,v^*}) and $(LR_{\ell,\underline{v}})$. Therefore $\sum_{k\epsilon T_1} y_k(v_k - b_k)$ is the same at the optimal solution for both $v = v^*$ and $v = \underline{v}$. Since $\underline{v}_1 > v_1^*$,

$$a_{ij} + \sum_{k\epsilon T_1} \underline{v}_k r_{ijk} \geq a_{ij} + \sum_{k\epsilon T_1} v_k^* r_{ijk} \quad \forall(i,j)\epsilon S \text{ and } (i,j)\epsilon \bar{S}$$

Therefore $Z(LR_{\ell,\underline{v}}) \geq Z(LR_{\ell,v}*)$. It follows that there exists an optimal solution to (D_ℓ) in which $v_1 \geq b_1$. The choice of k=1 being arbitrary, the same results hold for any value of k , $k\epsilon T$ and $y_k = 0$; which proves case (ii) of the Theorem.

It may be added that there is another possibility which complements case (ii) of Theorem 2, i.e., if $k\epsilon T$ and $y_k = 1$. We treat this possibility as a conjecture since a result similar to the one above could not be proved, as discussed now.

With $y_1 = 1$ and $v_1^* < b_1$, we observe from problem $(LR_{\ell,v}*)$ that for a solution vector X^* (with elements $x_{ij}*$) and Y^* (with elements $y_1*,\ldots,y_p^* | y_1* = 1$ and $y_2*,\ldots,y_p* = 0$ or 1) ,

$$Z(LR_{\ell,v}*) = \sum_{\substack{i\ j\\(i,j)\epsilon\bar{S}}}\sum x_{ij}*\left(a_{ij} + v_1^* r_{ij1} + \sum_{k>1} v_k* r_{ijk}\right)$$

$$+ \sum_{\substack{i\ j\\(i,j)\epsilon S}}\sum x_{ij}*\left(a_{ij} + v_1* r_{ij1} + \sum_{k>1} v_k* r_{ijk}\right)$$

$$- \sum_{k\epsilon\bar{T}} y_k*\left(v_k^* - b_k\right) - y_1\left(v_1* - b_1\right) - \sum_{\substack{k>1\\k\epsilon T}} y_k*\left(v_k* - b_k\right)$$

Since $v_1* < b_1$ and $y_1 = 1$ the term $-y_1(v_1* - b_1)$ is positive. If we raise v_1* to b_1 , say $\underline{v}_1$, the term $-y_1(\underline{v}_1 - b_1)$ is 0 .

The difference between $Z(LR_{\ell,v}*)$ and the objective function value of problem $(LR_{\ell,\underline{v}})$ with $X=X^*$ and $Y=Y^*$ is

$$= \sum_i\sum_j x_{ij}* v_1^* r_{ij1} + (b_1 - v_1*) - \sum_i\sum_j x_{ij}* b\ r_{ij1}$$

$$= (b_1 - v_1*) - \sum_i\sum_j x_{ij}* (b_1 - v_1*) r_{ij1} .$$

This difference can be either negative or positive, and so we cannot conclude that there is an optimal solution to problem (D_ℓ) in which

$v_1 \geq b_1$. We believe this conclusion to be false.

Theorem 3: Let (X^*, Y^*) solve problem (LR_v) for $v_k = b_k$ for all k . If (X^*, Y^*) is feasible for problem (P), there exists an optimal solution to problem (D) in which $v_k = b_k$ for all k .

Proof: In view of Theorem 1, there exists an optimal solution to (D) in which $v_k \geq b_k$ for all k , i.e., $v \geq b$. Let $\underline{v}$ be such an optimal v . We will show that $Z(LR_{\underline{v}}) \leq Z(LR_b)$, from which it follows that $v = b$ solves problem (D) .

Recall that

$$Z(LR_{\underline{v}}) = \min_{X,Y} \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k \underline{v}_k r_{ijk}\right) - \sum_k y_k \left(\underline{v}_k - b_k\right)$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad \forall j \qquad (2)$$

$$x_{ij} , y_k = 0 \text{ or } 1 \qquad \forall i,j,k \qquad (6)$$

Since $\underline{v} \geq b$, $y_k = 1 \ \forall k$ is an optimal choice.

Hence, $$Z(LR_{\underline{v}}) = \min_{X} \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k \underline{v}_k r_{ijk}\right) - \sum_k \left(\underline{v}_k - b_k\right)$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad \forall j \qquad (2)$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i,j \qquad (6a)$$

Now consider (LR_b) . Since $v = b$, the last term of the objective function drops out, and we have

$$Z(LR_b) = \min_{X,Y} \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k b_k r_{ijk}\right)$$

subject to (2) and (6)

$$= \underset{X}{\text{Min}} \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k b_k \, r_{ijk}\right)$$

subject to (2) and (6a)

$$= \sum_i \sum_j x_{ij}{}^* \left(a_{ij} + \sum_k b_k \, r_{ijk}\right)$$

where X^* with elements $x_{ij}{}^*$ is the minimizing solution vector which satisfies (2) and (6a) .

Now (X^*, Y^*) feasible for (P) implies that

$$\sum_i \sum_j x_{ij}{}^* \, r_{ijk} \leq y_k{}^* \leq 1 \qquad \forall k .$$

Hence, $\sum_k \left(\underline{v}_k - b_k\right) \sum_i \sum_j x_{ij}{}^* \, r_{ijk} \leq \sum_k \left(\underline{v}_k - b_k\right)$,

or $$\sum_k \left(\underline{v}_k - b_k\right) \sum_i \sum_j x_{ij}{}^* \, r_{ijk} - \sum_k \left(\underline{v}_k - b_k\right) \leq 0 \qquad (27)$$

Rewriting, $$Z(LR_{\underline{v}}) = \underset{X}{\text{Min}} \sum_i \sum_j x_{ij} \left[a_{ij} + \sum_k r_{ijk} \left(b_k + \left(\underline{v}_k - b_k\right)\right)\right] - \sum_k \left(\underline{v}_k - b_k\right)$$

subject to (2) and (6a)

$$= \underset{X}{\text{Min}} \left\{ \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k r_{ijk} \, b_k\right) + \sum_k \left(\underline{v}_k - b_k\right) \sum_i \sum_j x_{ij} \, r_{ijk} - \sum_k \left(\underline{v}_k - b_k\right) \right\}$$

subject to (2) and (6a)

$$= \sum_i \sum_j x_{ij}{}^* \left(a_{ij} + \sum_k r_{ijk} \, b_k\right) + \sum_k \left(\underline{v}_k - b_k\right) \sum_i \sum_j x_{ij}{}^* \, r_{ijk} - \sum_k \left(\underline{v}_k - b_k\right)$$

$$\leq \sum_i \sum_j x_{ij}{}^* \left(a_{ij} + \sum_k r_{ijk} \, b_k\right) = Z(LR_b)$$

by (27), or $Z(LR_v) \leq Z(LR_b)$; it follows that $v = b$ solves problem (D).

2.3 Relaxation (PR_ℓ)

Theorems 1 and 3 are useful in providing a choice of Lagrange multipliers as a starting point in solving a relaxation of problem (P) at the root node. Theorem 2, similar to Theorem 1, provides results for a partial solution of problem (P), i.e., at a node other than the root node where some of the x_{ij} and y_k have been fixed at 1 or 0 .

Theorem 1 is important in pointing out that a certain set of Lagrange multipliers v such that $v_k \geq b_k$ for all k would provide an optimal choice. Theorem 3 narrows this choice to $v_k = b_k$ for all k for a specific situation, i.e., when the resulting solution is feasible for problem (P).

Letting $v_k = b_k$ for all k , problem (LR_v) becomes:

$$(LR_b)\left\{\begin{array}{llll} \text{Minimize} & \sum_i \sum_j c_{ij}\, x_{ij} & & (28) \\ \text{subject to} & \sum_i x_{ij} = 1 & \forall j & (2) \\ & x_{ij} = 0 \text{ or } 1 & \forall i,j & (6a) \end{array}\right.$$

$$\text{where } c_{ij} = a_{ij} + \sum_k b_k\, r_{ijk} . \qquad (29)$$

Note that problem (LR_b) is very easy to solve; its optimal value is just the sum of the minimum (over i) c_{ij} for all j , i.e.,

$$Z(LR_b) = \sum_j \min_i \{c_{ij}\} \qquad (30)$$

We solve this problem as a starting point at the root node in our branch-and-bound procedure.

As we move to other nodes by fixing some of the variables, we must deal with problems having the form of problem (P_ℓ) instead of problem (P). The appropriate relaxation is then problem $(LR_{\ell,v})$, whose optimal value $Z(LR_{\ell,v})$ is the lower bound required at node ℓ. Our algorithm branches only on x_{ij} variables and uses the constraints (5') to fix appropriate y_k variables at values of 1. More precisely, if x_{ij} is fixed at 1 and $e_{ik} = 1$, then y_k must be 1 in every feasible completion of problem (P) so we can include the index k in T and fix y_k at 1. To account for the various possible combinations of i and j, we define

$$\begin{aligned} \alpha_{k\ell} &= 1 \text{ if } x_{ij}\, e_{ik} > 0 \text{ for any } (i,j)\varepsilon S\,, \\ &= 0 \text{ otherwise}. \end{aligned} \tag{31}$$

At any node ℓ then, y_k is fixed at 1 and $k\varepsilon T$ if $\alpha_{k\ell} = 1$.

There is another way in which it is appropriate to fix y_k at 1 at node ℓ. If the available choice of designs for some activity j requires the use of facility k, then y_k may be set to 1. Formally, define

$$W = \{j \mid (i,j)\varepsilon S \text{ and } x_{ij} = 1 \text{ for some } i\} \tag{32}$$

and its complement $\bar{W}$. Then define

$$\begin{aligned} \beta_{k\ell} &= 1 \text{ if } \sum_{j\varepsilon\bar{W}} \min_{\substack{i \\ (i,j)\varepsilon\bar{S}}} d_{ijk} > 0\,, \\ &= 0 \text{ otherwise}. \end{aligned} \tag{33}$$

Then y_k is fixed at 1 and $k\varepsilon T$ if $\beta_{k\ell} = 1$. It is convenient to combine these two notations in forcing y_k to 1. Define

$$\delta_{k\ell} = \text{Max}\{\alpha_{k\ell}\ ,\ \beta_{k\ell}\} \tag{34}$$

so y_k is fixed at 1 and $k\epsilon T$ if $\delta_{k\ell} = 1$.

To return to the relaxation, problem $(LR_{\ell,v})$, we must make a choice of the vector v of Lagrange multipliers. Of course, we would like to use an optimal choice, i.e., a vector v that solves problem (D_ℓ) . Recall, however, that Theorem 2 did not provide us any useful information about the optimal value of v_k if $k\epsilon T$ and $y_k = 1$. To simplify our approach and have recourse to the results of Theorems 1 and 3, we choose $v_k = 0$ if $k\epsilon T$ and $y_k = 1$. Note that there are no $k\epsilon T$ such that $y_k = 0$ because of practical considerations and because our branching rule only results in fixing y_k values at 1 . Problem $(LR_{\ell,v})$ now takes the form

$$(LR_{\ell,\bar{v}}) \begin{cases} \text{Minimize} & \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_{k\epsilon\bar{T}} v_k r_{ijk}\right) - \sum_{k\epsilon\bar{T}} y_k \left(v_k - b_k\right) + \sum_{k\epsilon T} b_k & (35) \\ \text{subject to} & \sum_i x_{ij} = 1\ , \qquad \forall j & (6) \\ & x_{ij},\ y_k = 0 \text{ or } 1 \ \text{ for all } (i,j)\epsilon\bar{S}\ ,\ k\epsilon T\ . & (23) \end{cases}$$

Note that in this problem $(LR_{\ell,\bar{v}})$, $\bar{v}_k = 0$ if $k\epsilon T$. Also note how closely it resembles problem (LR_v), the relaxation at the root node. As in that case, we would like the lower bound $Z(LR_{\ell,\bar{v}})$ to be as large as possible, i.e., we seek $\bar{v}$ to

$$(D_{\bar{\ell}}) \begin{cases} \text{Maximize} \left[Z(LR_{\ell,\bar{v}})\right] & (36) \\ \bar{v} \geq 0 & \end{cases}$$

Because of the close similarity of problems $(LR_{\ell,\bar{v}})$ and (LR_v), it is possible to obtain results about problem $(D_{\bar{\ell}})$ that are analogous to those obtained about problem (D). We state these results as Theorems 4 and 5. Their proofs are omitted because they follow precisely the proofs of Theorems 1 and 3, respectively, and their validity follows from the fact that problem $(LR_{\ell,\bar{v}})$ is essentially the same as problem (LR_v) but involves only the free variables.

Theorem 4: There exists an optimal solution to problem $(D_{\bar{\ell}})$ in which $v_k \geq b_k$ for all $k \in \bar{T}$.

Theorem 5: Let (X^*, Y^*) solve problem $(LR_{\ell,\bar{v}})$ for $v_k = b_k$ for all $k \in \bar{T}$. If (X^*, Y^*) satisfies (5') for all $k \in \bar{T}$, there exists an optimal solution to problem $(D_{\bar{\ell}})$ in which $v_k = b_k$ for all $k \in \bar{T}$.

Just as Theorems 1 and 3 motivated us to use the relaxation problem (LR_b) to obtain our lower bound at node 1 , Theorems 4 and 5 motivate us to set $v_k = b_k$ for all $k \in \bar{T}$ in relaxation problem $(LR_{\ell,\bar{v}})$ to obtain our lower bound at node ℓ . With this specification, problem $(LR_{\ell,\bar{v}})$ becomes

$$(PR_\ell) \begin{cases} \text{Minimize} & \sum_i \sum_j c_{ij\ell}\, x_{ij} + FC_\ell & & (37) \\ \text{subject to} & \sum_i x_{ij} = 1 & \forall j & (2) \\ & x_{ij} = 0 \text{ or } 1 & \forall (i,j) \in \bar{S} \ , & (23a) \end{cases}$$

where

$$c_{ij\ell} = a_{ij} + \sum_{k \in \bar{T}} b_k\, r_{ijk}$$

$$= a_{ij} + \sum_{k=1}^{p} b_k \left(1 - \delta_{k\ell}\right) r_{ijk} \qquad (38)$$

and the fixed cost FC_ℓ is given by

$$FC_\ell = \sum_{k \in T} b_k = \sum_{k=1}^{p} \delta_{k\ell}\, b_k \ . \qquad (39)$$

This specific relaxation, problem (PR_ℓ), is of the same form as problem (LR_b) and is equally easy to solve in one pass. Its optimal value $Z(PR_\ell)$ serves as the lower bound at node ℓ . Note that for $\ell=1$, problem (PR_ℓ) is the same as problem (LR_b) .

It is clear that setting each Lagrange multiplier v_k to b_k for $k\epsilon\bar{T}$ and to 0 for $k\epsilon T$ is not generally optimal in terms of achieving the tightest lower bound (except as per Theorem 3). But it provides a good starting point in seeking an optimal vector v and it provides an easily calculated lower bound at each node of our branch-and-bound procedure. The question of how to improve upon this choice of multiplier values will be discussed in Chapter 6.

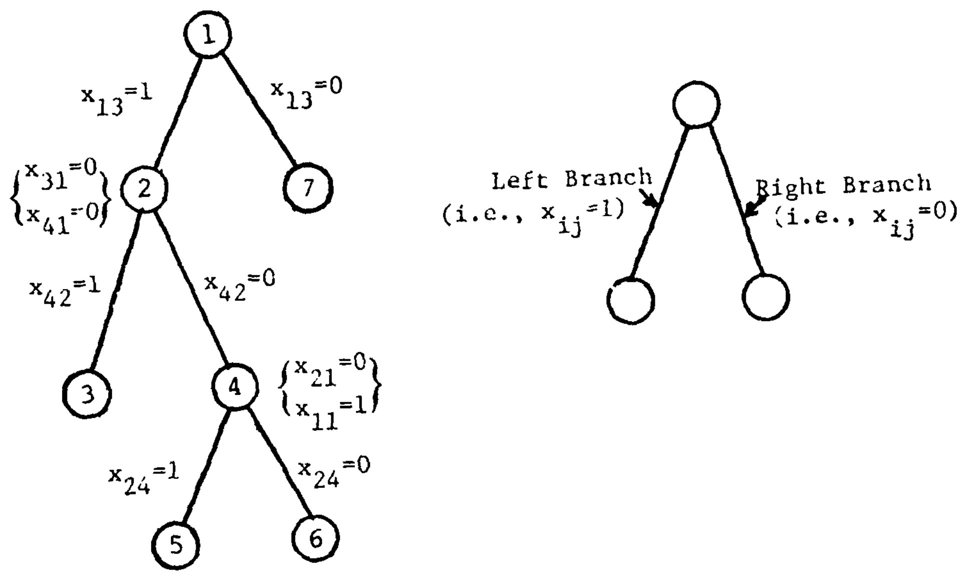
3. METHODOLOGY FRAMEWORK

The branch-and-bound procedure/methodology developed to solve problem (P) uses Lagrangian relaxation (PR_ℓ) as a basic step. The branching rule dictates which x_{ij} variable to branch on at each node. In addition, there are certain rules (e.g., the capacity rule and the bounding rule) which contribute, significantly, in improving the overall efficiency of the procedure.

Some basic terms such as fixed and free variables, partial solution and its completion were introduced in the previous chapter. This chapter first provides a preliminary discussion of the branch-and-bound methodology, [Geoffrion (1967), and Geoffrion and Marsten (1972)]. Representation and storage of the x_{ij} variables for branching and backtracking is described in order to provide continuity and consistency with the computer program covered in Chapter 4. This is followed by a description of the major components of the branch-and-bound methodology.

Branching and backtracking is done on the x_{ij} variables. The branching commences by fixing the x_{ij} variable (selected by the branching rule) to 1 and moving to the left branch node. When backtracking, we fix the corresponding x_{ij} variable at 0 and move to the right branch node (if the right branch node has not already been explored). An x_{ij} variable can also be fixed at 0 or 1 by rules other than the branching rule. The capacity rule and the bounding rule are two such rules employed in our methodology.

Figure 4a shows a branch-and-bound tree. The x_{ij} variables fixed at 0 or 1 at any node due to rules other than the branching rule are shown in parenthesis at the appropriate node.



$x_{13}=1$
$x_{13}=0$
$\begin{Bmatrix} x_{31}=0 \\ x_{41}=0 \end{Bmatrix}$
$x_{42}=1$
$x_{42}=0$
$\begin{Bmatrix} x_{21}=0 \\ x_{11}=1 \end{Bmatrix}$
$x_{24}=1$
$x_{24}=0$
Left Branch (i.e., $x_{ij}=1$)
Right Branch (i.e., $x_{ij}=0$)

Node ① is the root node and also the parent node for nodes ② and ⑦

Node ② is the parent node for nodes ③ and ④, etc.

Figure 4a.

A branch-and-bound tree illustration

Node (ℓ)	Partial Solution (S_ℓ)
1	ϕ
2	{103, - 301, - 401}
3	{103, - 301, - 401, 402}
4	{103, - 301, - 401, - 402, - 201, $\underline{101}$}
5	{103, - 301, - 401, - 402, - 201, $\underline{101}$, 204}
6	{103, - 301, - 401, - 402, - 201, $\underline{101}$,-204}
7	{- 103}

Figure 4b.

Partial solutions for the above illustration (Figure 4a)

For problem (PR_ℓ), a partial solution corresponding to set S at node ℓ, i.e, S_ℓ contains x_{ij} variables assigned values of 1 or 0. For simplicity in the computer program, an x_{ij} variable fixed at 1 is represented as $(100\, i + j)$, whereas an x_{ij} variable fixed at 0 as $-(100\, i + j)$, e.g., $x_{32} = 1$ and $x_{32} = 0$ are represented as 302 and -302 respectively. Since branching is done on x_{ij} variables, it is necessary to make a distinction between x_{ij} variables fixed at 1 due to the branching rule and those fixed at 1 due to the other rules. We make this distinction by underlining the positive number to represent an x_{ij} fixed at 1 due to the other rules. For example, 204, -301, $\underline{103}$ represent, respectively, $x_{24} = 1$ due to the branching rule, $x_{31} = 0$ due to the branching rule or any other rule, and $x_{13} = 1$ due to a rule other than the branching rule.

Figure 4b shows the partial solutions S_ℓ of the branch-and-bound tree in Figure 4a.

Implicit enumeration involves generating a sequence of partial solutions and simultaneously considering all completions of each. For our minimization problem, we start with an initial solution having a very large value (infinity) as an initial upper bound. As the computations proceed, feasible solutions (those satisfying the capacity constraints) are discovered from time to time, and the best one yet found is retained as an incumbent solution with the corresponding value as the best upper bound. It may happen that for a given partial solution S_ℓ we can determine a best completion of S_ℓ, i.e., a feasible completion that minimizes the objective function value among all feasible completions of S_ℓ. If such a best feasible completion is better than the best upper bound, then it replaces the latter. Or we may be able to determine that S_ℓ has no feasible completion better than the incumbent. In either case, we can fathom S_ℓ. (Various situations of fathoming and backtracking in our branch-and-bound procedure are described in the following discussion.) All completions of a fathomed partial solution S_ℓ have been implicitly enumerated in the sense that they can be excluded from further consideration (with the exception of the relevant best feasible solution of S_ℓ if it has been retained as the best upper bound).

In our branch-and-bound procedure, at any given node where we can fathom S_ℓ, we backtrack to the parent node and move to the right-hand branch (if that branch has not already been explored) by fixing the appropriate x_{ij} variable at 0. However, if the right-hand branch has already been explored, we continue backtracking to a parent node where we can move to a right-hand branch. For example, in Figure 4a, when backtracking from node 3, we move to the parent node 2, and to the right to node 4 by setting $x_{42} = 0$. However, when backtracking from node 6, we move back to node 4, then back to node 2, then back to node 1, and to the right to node 7 by setting $x_{13} = 0$.

On the other hand, if the partial solution S_ℓ cannot be fathomed, we branch to the left and augment S_ℓ by fixing a free variable x_{ij} at 1 (based on the branching rule), and then we try to fathom the resulting partial solution. In addition to the one variable selected by the branching rule, some other free x_{ij} variables can also be fixed at 0 or 1 according to the application of rules other than the branching rule. Note that this can also happen when backtracking, i.e., when S_ℓ has been fathomed and we backtrack and move to the right by setting the appropriate x_{ij} variable to 0.

Let us consider examples of both situations, i.e., when S_ℓ has not been fathomed and when S_ℓ has been fathomed. In Figure 4a we cannot fathom S_1 (i.e., S at node 1), so we move to node 2 by augmenting S_1 by fixing $x_{13} = 1$ based on the branching rule, and by fixing $x_{31} = 0$ and $x_{41} = 0$ based on the application of the other rules. Similarly, we move from node 2 to node 3 by augmenting S_2 by fixing $x_{42} = 1$. As an example of backtracking, wher we fathom S_3 , we move back to the parent node 2, and to the right to node 4, getting a new partial solution S_4 by replacing $x_{42} = 1$ with $x_{42} = 0$, and further augmenting it by fixing $x_{21} = 0$ and $x_{11} = 1$ based on the application of the other rules.

Computationally, the storage and update of partial solution S_ℓ is easily accomplished by considering Figure 4b. If, at a given node, the partial solution S_ℓ has not been fathomed, e.g., at node 4, determine the next branching variable by using the branching rule, i.e., x_{24} , and augment S_4 by adding 204 as the last entry. Also, augment S_4 with any other free x_{ij} variables, if appropriate, depending on the application of the other rules. Now, consider the case where the partial solution S_ℓ has been fathomed, e.g., at node 6, and we backtrack; starting with the last entry in S_ℓ . we consider one entry at a time, going backwards, until we find a positive number which is not underlined. In our example, it is 103. In other words, we must branch to the right by fixing $X_{13} = 0$, i.e., we replace 103 with - 103 and we are at node 7. Should we find that we have no positive number, the procedure terminates since we are back at the root node and the right branch has already been explored. This happens when backtracking from node 7.

In the branch-and-bound procedure we generate a sequence of partial solutions as we move from one node to another. This sequence is non-redundant in the sense that no completion of a partial solution ever duplicates a completion of a previous partial solution that has been fathomed.

Since one of the x_{ij} values, for each j, must be 1, a total of $(2m-1)^n$ nodes are theoretically possible for complete enumeration. However, most of the solutions may be infeasible because of the capacity constraints. The branch-and-bound procedure, through a judicious choice of branching variables, and elimination of certain infeasible and non-optimal assignments through various rules, turns out to be a practical and computationally efficent algorithm. The various components of this procedure are described next. Detailed procedural steps and the solution of a test problem will be covered in Chapter 4.

3.1 Bounds

3.1.1 Lower Bound

At a given node ℓ in the branch-and-bound tree, a lower bound (LOWB) is obtained by solving relaxed problem (PR_ℓ).

$$\text{LOWB} = Z(PR_\ell) \tag{40}$$

Recall that problem (PR_ℓ) is very easy to solve by considering the minimum $c_{ij\ell}$ over those j's for which x_{ij} is not fixed at 1, i.e., $j \varepsilon \bar{W}$, where $\bar{W}$ is the complement of W defined by expression (32).

$$Z(PR_\ell) = \sum_{j \varepsilon W} c_{ij\ell} + \sum_{j \varepsilon \bar{W}} \min_{\substack{i \\ (i,j)\varepsilon \bar{S}}} c_{ij\ell} + FC_\ell \,, \tag{41}$$

where $c_{ij\ell}$ is given by expression (38), i.e.,

$$c_{ij\ell} = a_{ij} + \sum_k b_k (1 - \delta_{k\ell})\, r_{ijk} \,, \tag{38}$$

and the fixed cost (FC_ℓ) is given by expression (39), i.e.,

$$FC_\ell = \sum_k \delta_{k\ell}\, b_k \,, \tag{39}$$

where $\delta_{k\ell}$ is given by expression (34).

Note that if none of the x_{ij} variables is fixed at 1 , as is generally the case at the root node, then all $\delta_{kl} = 0$, and, therefore, $FC_l = 0$, and $c_{ijl} = a_{ij} + \sum_{k=1}^{p} b_k r_{ijk}$. $Z(PR_1)$ is, then, simply the middle part of expression (41). We use the term "generally" because it is possible that the capacity rule could force certain x_{ij} variables to 1 (or 0) at the root node, prior to solving the relaxed problem (PR_1) .

3.1.2 Upper Bound

At any given node ℓ , let $X = \{x_{ij}\}$ represent the solution of problem (PR_ℓ) . If this solution is feasible for problem (P), i.e., if X satisfies the capacity constraints (5) or (5')

$$\sum_{\substack{i\ j \\ x_{ij}\varepsilon X}} \sum d_{ijk}\, x_{ij} \le s_k\, y_k \qquad \forall k , \tag{42}$$

$$\text{where } y_k = 1 \text{ if } \sum_{\substack{i\ j \\ x_{ij}\varepsilon X}} \sum d_{ijk}\, x_{ij} > 0 , \qquad = 0 \text{ otherwise} , \tag{43}$$

then the value of problem (P) corresponding to this solution gives an <u>upper bound</u> (UPB):

$$UPB = \sum_{\substack{i\ j \\ x_{ij}\varepsilon X}} \sum a_{ij}\, x_{ij} + \sum_k b_k y_k , \tag{44}$$

where y_k is defined by (43) .

3.1.3 Best Upper Bound

A current lowest upper bound is retained as the <u>best upper bound</u> (BUB), the corresponding solution X representing the incumbent solution.

The branch-and-bound procedure is initiated by assuming a very large value as the best upper bound, and is replaced by better (lower) values as the procedure continues.

A positive fractional value ε can be specified if a sub-optimal solution is acceptable. For example, for $\varepsilon = 0.001$, the resulting solution value is guaranteed to be within 0.1 percent of the optimal solution value. When ε is non-zero, the adjusted best upper bound (BUBS) is defined as:

$$\text{BUBS} = \text{BUB}/(1 + \varepsilon) \ . \qquad (45)$$

Obviously when $\varepsilon = 0$, BUBS = BUB .

3.2 Facility Usage Rule

This rule is used to identify facilities forced into usage at a given node ℓ and hence fix corresponding free variables y_k at 1 .

For a partial solution S_ℓ, define

$$\begin{aligned} \bar{d}_{jk\ell} &= d_{ijk} \quad \text{if} \quad j \varepsilon W , \\ &= \min_{\substack{i \\ (i,j)\varepsilon\bar{S}}} d_{ijk} \quad \text{if} \quad j \varepsilon \bar{W} . \end{aligned} \qquad (46)$$

The facility usage rule states that for any facility k , where y_k is not already fixed at 1 , if $\sum_j \bar{d}_{jk\ell} > 0$, then facility k is forced into usage and, therefore, y_k should be fixed at 1 .

This rule is applied at every node prior to applying the capacity rule. In other words, this rule is applicable to capacitated as well as uncapacitated problems.

3.3 Capacity Rule

This rule is designed to "exclude" infeasible assignments prior to solving the relaxed problem (PR_ℓ) . This is done by exploiting the relationship between the capacities required (d_{ijk}) and the capacities available (s_k) for a given partial solution of problem (P).

The capacity rule states that for a facility k and an activity j, "exclude" a free x_{ij} variable (i.e., fix it at 0) for which

$$(d_{ijk} - \bar{d}_{jk\ell}) > (s_k - \sum_j \bar{d}_{jk\ell}) , \qquad (i,j)\varepsilon\bar{S} \qquad (47)$$

where $\bar{d}_{jk\ell}$ is defined by expression (46). The right-hand side of this inequality (47), when positive, represents the available capacity at facility k. The left-hand side shows, for a given j, the difference between a d_{ijk} corresponding to a free x_{ij} variable and $\bar{d}_{jk\ell}$. If, for a specific d_{ijk}, this difference is more than the available capacity, the corresponding free x_{ij} variable, if fixed at 1, would result in an infeasible solution. Thus, by looking ahead, we can exclude such a free x_{ij} variable by assigning it a value of 0.

Note that if the right-hand side of expression (47) is negative, then any completion of such a partial solution will be infeasible and we backtrack in our branch-and-bound procedure.

The capacity rule is applied to all the facilities by considering one facility at a time. The cycle of examining all the facilities continues until no more assignments can be excluded. During the course of application of this rule, if all but one of the free x_{ij} variables have been excluded (fixed at 0) for a given j, then that particular x_{ij} variable is fixed at 1 because of constraints (2), i.e., each activity j must be assigned to one and only one design i. The partial solution is updated accordingly to reflect the x_{ij} variables fixed at 0 or 1 due to the application of the capacity rule.

The capacity constraints for an uncapacitated problem are not active. Hence, the capacity rule is useful only for capacitated problems.

3.4 Branching Rule

This rule provides the choice of the x_{ij} variables on which to branch. If the partial solution at a given node ℓ is not fathomed, we branch further by fixing a free x_{ij} variable at 1 and moving to the left branch node.

According to the branching rule the choice of the branching variable depends on the $c_{ij\ell}$ values and is such that the corresponding x_{ij} , if perturbed, has the maximum impact on the optimal value of problem (PR_ℓ) .

For a given j , define $c_{i_1 j\ell}$, the minimum permissible $c_{ij\ell}$, and $c_{i_2 j\ell}$, the second smallest permissible $c_{ij\ell}$, i.e.,

$$c_{i_1 j\ell} = \min_{i} c_{ij\ell} \quad \text{for} \quad j\varepsilon\bar{W} \quad \text{and} \quad (i,j)\varepsilon\bar{S} \tag{48}$$

$$\text{and} \quad c_{i_2 j\ell} = \min_{\substack{i \\ i \neq i_1}} c_{ij\ell} \quad \text{for} \quad j\varepsilon\bar{W} \quad \text{and} \quad (i,j)\varepsilon\bar{S} \tag{49}$$

$$\text{For each } j\varepsilon\bar{W}\text{, define } D_{j\ell} = c_{i_2 j\ell} - c_{i_1 j\ell} . \tag{50}$$

Our branching rule states that a free x_{ij} variable corresponding to $c_{i_1 j\ell}$ such that $D_{j\ell}$ is maximized over all j , is selected as the next branching variable and assigned a value of 1.

3.5 Bounding Rule

This rule is designed to "exclude" certain non-optimal assignments. These assignments cannot lead to an optimal solution as we branch from one node to the next left branch node.

The bounding rule states that a free x_{ij} variable should be excluded (by assigning it the value 0) for which

$$(c_{ij\ell} - c_{i_L j\ell}) > (BUBS - LOWB) \quad \text{for} \quad j\varepsilon\bar{W} \quad \text{and} \quad (i,j)\varepsilon\bar{S} \qquad (51)$$

where $c_{i_L j\ell}$, BUBS, and LOWB are given by expressions (48), (45), and (40), respectively.

Thus, by looking ahead, we exclude those assignments which will provide lower bounds higher than BUBS.

The bounding rule is applied to each $j\varepsilon\bar{W}$ just prior to selecting the x_{ij} variable for branching to the left.

As in the case of the capacity rule, if the bounding rule results in excluding (fixing at 0) all but one of the free x_{ij} variables for a given $j\varepsilon\bar{W}$, then that particular x_{ij} variable is fixed at 1 . Also the partial solution is updated accordingly to reflect the x_{ij} variables fixed at 0 or 1 due to the application of the bounding rule.

3.6 Backtracking Rules

If a partial solution at a given node has been fathomed, we backtrack. The backtracking rules are typical of a branch-and-bound procedure. In addition, the application of the capacity rule and the bounding rule can lead to backtracking. The criteria for backtracking include the following.

(a) When applying the capacity rule, if the available capacity given by the right-hand side of inequality (47) is negative, i.e., $(s_k - \sum_j \bar{d}_{jk\ell}) < 0$, then backtrack.

(b) If LOWB $\geq$ BUBS, then backtrack. Otherwise compute UPB if the solution is feasible in problem (P). Then update BUB and BUBS if UPB < BUB; and backtrack if LOWB = BUBS.

(c) If further branching is not possible, then backtrack. This can happen due to the capacity rule, the bounding rule, or the branching rule if the updated partial solution is such that no further branching is possible, i.e., x_{ij} variables are fixed at 1 for all j , or equivalently, $\bar{W} = \phi$.

When any of the backtracking criteria apply, we backtrack to the parent node and move to the right branch node (if the right branch has not already been explored) by fixing the appropriate x_{ij} variable at 0 . If the right branch has already been explored, we continue backtracking to a parent node where we can move to a right branch node. The branch-and-bound procedure terminates when we backtrack to the root node and find that the right branch node has already been explored.

4. COMPUTATIONAL STEPS AND THE COMPUTER PROGRAM

A computer program called ZIPCAP (an acronym for Zero-one Integer Program for multiactivity multifacility Capacity-constrained Assignment Problems) implementing the branch-and-bound methodology has been developed.

Detailed procedural steps and guidelines to use the computer program are described in a separate document [Chhabra and Soland (1980)] titled "Program Description and User's Guide for ZIPCAP--a Zero-one Integer Program to solve multiactivity multifacility Capacity-constrained Assignment Problems." Specifically, the document includes:

- Problem formulation (P) and potential areas of application
- Overall flow diagram and detailed procedural steps for the computer program
- Program listing and dictionary of the symbolic names. The listing includes extensive use of comment cards to explain various computational steps.
- User information including
  - schematic diagram of the deck structure,
  - detailed instructions for the job control (JCL) cards, program parameter card, program options card, and the various other input data cards.
- Three test problems to demonstrate the use of the program. The display includes coded input and annotated outputs reflecting the use of selected program options.

As mentioned earlier, ZIPCAP is primarily designed for capacitated problems. However, uncapacitated problems can be solved as a special case, and this is demonstrated by including an uncapacitated test problem.

Because of the extensive coverage of the program description and user guidelines in the above document, this chapter provides only an overview of the computer program, including an overall flow diagram, and a summary of the program options, in order to provide continuity in this document. In addition, a step-by-step description of a test problem is presented to demonstrate the use of the various components of the branch-and-bound methodology. The computer printout showing step-by-step details is obtained by use of one of the program options. The use of this option to display detailed steps in this document, in fact, complements the use of the various options demonstrated in the other document.

4.1 The Program

Figure 5 presents a simplified flow diagram of the branch-and-bound procedure. The major computational steps for the computer program are numbered in circles. These steps are essentially based on the methodology components described in the previous chapter. A step-by-step description has been included in the other document [Chhabra and Soland (1980)].

The computer program ZIPCAP is written in FORTRAN IV, and has been developed and tested on an IBM 3031 at the George Washington University. The program, comprising about 480 lines is currently dimensioned for a maximum problem size of 35 designs (m), 35 activities (n) and 30 facilities (p). The program size to execute a problem has two components: one, due to the program itself, comprising 173 K bytes, and the other dependent on the dimensions of the arrays given by the following functional relationship.

$$f(m,n,p) = 4[(p+4)mn + (m+5)p+9n] \text{ bytes}$$

The computer program listed in the other document has since been further improved. The basic improvement has been the addition of the facility usage rule. This rule, as described in Chapter 3, is applied both to capacitated and uncapacitated problems just before the application of the capacity rule. For completeness of this document, a revised program listing is included in Appendix A. It may be mentioned that

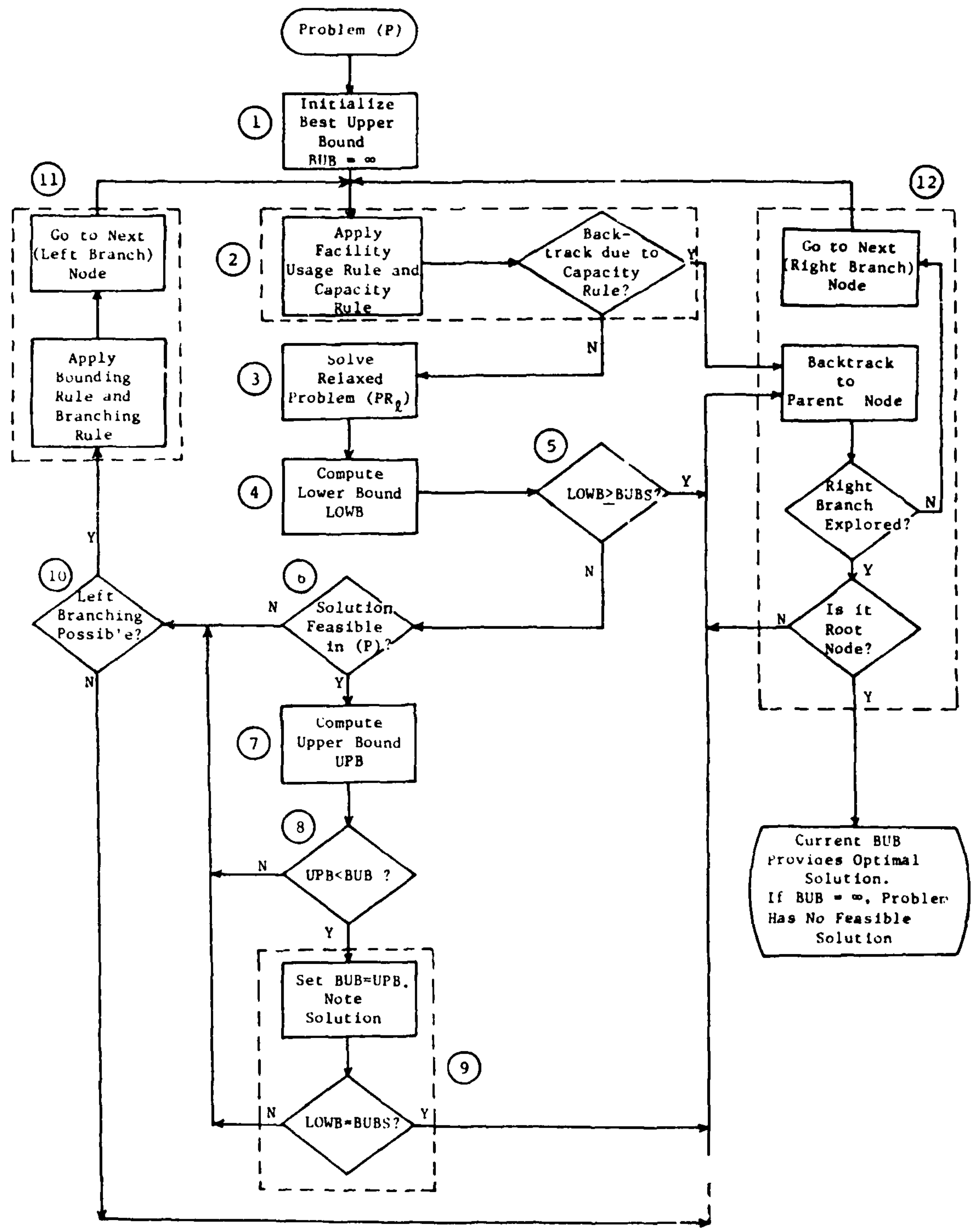

Problem (P)
1
Initialize Best Upper Bound BUB = ∞
11
Go to Next (Left Branch) Node
Apply Bounding Rule and Branching Rule
2
Apply Facility Usage Rule and Capacity Rule
Back-track due to Capacity Rule?
12
Go to Next (Right Branch) Node
Backtrack to Parent Node
Right Branch Explored?
Is it Root Node?
3
Solve Relaxed Problem (PR_ℓ)
4
Compute Lower Bound LOWB
5
LOWB≥BUBS?
6
Solution Feasible in (P)?
10
Left Branching Possible?
7
Compute Upper Bound UPB
8
UPB<BUB ?
9
Set BUB=UPB. Note Solution
LOWB=BUBS?
Current BUB Provides Optimal Solution. If BUB = ∞, Problem Has No Feasible Solution
Y
N

Figure 5.
Simplified flow diagram for the branch-and-bound procedure

the revised program solves the test problems included in the other document more efficiently -- in less time and in fewer nodes (with an average reduction in nodes of 31 percent). The improvement in efficiency seems to result from the "multiplicative" effect of the various rules. Another improvement made is that the computer printout always displays the node number (IBNOD) at which the best upper bound changes (improves) and the corresponding values of the best upper bound (BUB) and the adjusted best upper bound (BLBS).

ZIPCAP provides numerous options to the program user. These options, described in the other document, are summarized in Table 3.

Option ICAPR, the capacity rule, is automatically skipped by the program when solving an uncapacitated problem. Option ISTEP, the intermediate steps' listing, even when skipped, provides information on the total number of nodes explored. A summary listing provides necessary information to construct the branch-and-bound tree, whereas a detailed listing of the intermediate steps is useful when changing or debugging the program.

Option EPS, the optimal/suboptimal solution, provides the flexibility of obtaining a suboptimal value guaranteed to be within a specified fraction of the optimal value. The resulting solution may be suboptimal but could provide a considerable saving in terms of exploring fewer nodes in comparison to those necessary for obtaining an optimal solution.

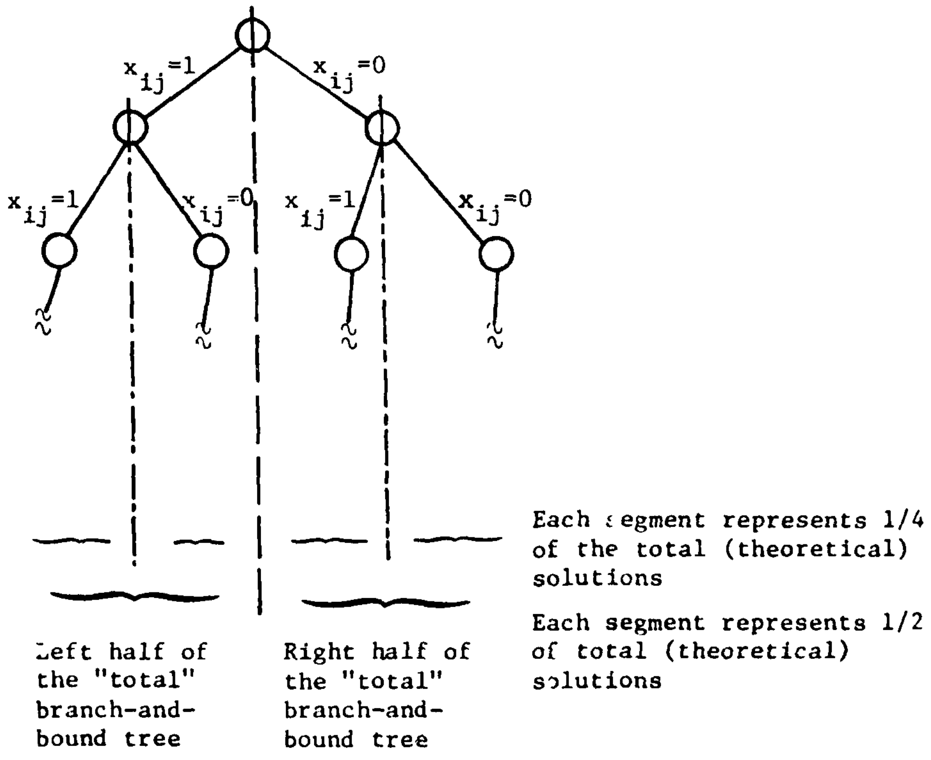
Option ET, by providing important information at a specified elapsed time, is useful in a situation where the total time allocated to solve a problem may not be sufficient and the program terminates before verifying an optimal solution. The information provided by this option includes an updated partial solution showing the x_{ij} variables fixed at 0 or 1, at the current node being explored at the specified time ET. By looking at the first few variables displayed in the partial solution of the current node, it is possible to assess the extent of the branch-and-bound tree explored until time ET. For

Table 3

SUMMARY OF ZIPCAP OPTIONS

	Option Name		Alternatives Available to User
1.	IINPT	-- Input Listing	. List input data . Skip this option
2.	ICAPR	-- Capacity Rule	. Use capacity rule . Skip this option
3.	ISTEP	-- Intermediate Steps Listing	. Skip listing of intermediate steps . Provide a summary of intermediate steps . Provide detailed intermediate steps
4.	IUNCAP	-- Capacitated/Uncapacitated Problem	. Solve a capacitated problem . Solve an uncapacitated problem
5.	EPS	-- Optimal/Suboptimal Solution	. Optimal solution . Suboptimal solution acceptable within a specified fractional value (epsilon)
6.	ET	-- Information at a specified Elapsed Time	. Provide the following information at elapsed time ET: - Best upper bound, corresponding solution, and the node at which found - node being explored and detailed steps for that node . Skip this option

example, in view of the terminology in Figure 4b (Chapter 3), if, at an arbitrary node, the first term of the partial solution is positive, i.e., the x_{ij} variable has value 1, then we are still in the left half of the total branch-and-bound tree. If the first term is negative, i.e., the x_{ij} variable has value 0, then we are in the right half of the total branch-and-bound tree and have explored half of the total (theoretical) solutions corresponding to the left half of the tree. If the first two terms are negative, i.e., the first two x_{ij} variables have value 0, then one quarter of the total (theoretical) solutions remain to be explored, since we are in the next right half of the right half of the total branch-and-bound tree, as illustrated in Figure 6.



$x_{ij}=1$
$x_{ij}=0$
$x_{ij}=1$
$x_{ij}=0$
$x_{ij}=1$
$x_{ij}=0$
Each segment represents 1/4 of the total (theoretical) solutions
Each segment represents 1/2 of total (theoretical) solutions
Left half of the "total" branch-and-bound tree
Right half of the "total" branch-and-bound tree

Figure 6

Illustration for estimating the extent of the branch-and-bound tree explored

Recall from Chapter 3, that a total of $(2m-1)^n$ nodes are theoretically possible. Thus, if the first g $[g \leq (m-1)n]$ terms at an arbitrary node are negative, then theoretically about $[(2m-1)^n/_2g]$ nodes remain to be explored.

4.2 An Illustrative Example

We consider a capacitated test problem with five designs (m), four activities (n), and eight facilities (p) to demonstrate the use of the branch-and-bound procedure and the computer program.

The computer printout for this problem showing step-by-step details for a couple of nodes is presented in Appendix B.

As shown in the beginning of the printout, the options selected are:

. IINPT = 1, i.e., list the input data
. ICAPR = 1, i.e., use the capacity rule
. ISTEP = 2, i.e., list detailed intermediate steps
. IUNCAP = 1, since this is a capacitated problem
. EPS = 0.0 implying that an optimal solution is desired
. ET = 0.0 since a detailed listing of intermediate steps will be available.

Following the listing of the options, input data listed for the problem include variable costs a_{ij}, fixed costs b_k, available capacities s_k, and capacities required d_{ijk}. The e_{ik} values are generated by the computer program.

The computer program follows the procedural steps marked in the flow diagram presented in Figure 5. These steps, along with the relevant terminology used in the computer printout, are described below for a couple of nodes, followed by a complete branch-and-bound tree for this problem. As mentioned earlier, a dictionary of the symbolic names used in the computer program is included in the other document.

Node 1

Step 1: Initialize.

Initialize BUB = 9999999.0, and since EPS = 0.0, BUBS = BUB. Also $S = \phi$ and $W = \phi$. In the computer printout, vector FIX(J) represents the set W, and matrix CX(I,J) represents both, fixed and free x_{ij} variables. In the CX(I,J) matrix, an x_{ij} variable fixed at 1 or 0 is represented as 1 or 2, respectively, and a free x_{ij} variable is represented by the value 0. Initially, all the x_{ij} variables are free as shown by matrix CX(I,J) in the printout.

Step 2: Apply the facility usage rule and the capacity rule for k=1,2,...,8 .

In the printout, MIND(J) represents $\bar{d}_{jk\ell}$ defined by expression (46), and MINSD represents $\sum_j \bar{d}_{jk\ell}$. As shown in the printout, MINSD is 0 for k=1,2,...,8 , and so the facility usage rule does not force any facilities into usage; and as shown by matrix CX(I,J) for k=1,2,...8 , the capacity rule does not fix any x_{ij} variables.

Step 3: Solve the relaxed problem (PR_1) .

In the printout FLB(K) represents $\delta_{k\ell}$, given by expression (34), for computing FC_ℓ , and C(I,J) represents $c_{ij\ell}$ defined by expression (38). Being at the root node, $\ell = 1$. Further the solution of problem (PR_ℓ) , i.e., $X = \{x_{ij}\}$ is shown in the printout by SOLX(J) which for (PR_1) is $X = \{x_{41} = x_{42} = x_{23} = x_{44} = 1\}$.

Step 4: Compute the lower bound.

The expressions (40) and (41), i.e.,

$$\text{LOWB} = Z(PR_\ell) \tag{40}$$

$$= \sum_{j\in W} c_{ij\ell} + \sum_{j\in \bar{W}} \min_{\substack{i \\ (i,j)\in\bar{S}}} c_{ij\ell} + FC_\ell \tag{41}$$

are represented in the printout as

$$\text{LOWB} = \text{MINSC} + \text{FC}$$
$$= 729839.3125 + 0 = 729839.3125$$

Step 5: Compare LOWB with BUBS.

Since LOWB < BUBS, go to Step 6

Step 6: Check if solution X is feasible in problem (P), i.e., expression (42) is satisfied.

$$\sum_{\substack{i\ j \\ x_{ij}\in X}} \sum d_{ijk}\, x_{ij} \leq s_k y_k \qquad \forall k \tag{42}$$

In the printout, NSUMD represents the left-hand side of this inequality, and for each k , the capacity constraints are satisfied.

Step 7: Compute the upper bound.

UPB is given by expression (44), i.e.,

$$\text{UPB} = \sum_{\substack{i\ j \\ x_{ij}\in X}} \sum a_{ij}\, x_{ij} + \sum_k b_k y_k \tag{44}$$

In the printout, the corresponding expression is represented as

$$\text{UPB} = \text{NSUMA} + \text{FCUB}$$
$$= 678{,}502 + 101{,}000 = 779502.0 .$$

Step 8: Compare UPB with BUB.

Since UPB < BUB, go to Step 9.

Step 9: Set BUB = 779502.0 . Since EPS = 0.0, BUBS = BUB.

Since LOWB < BUBS, go to Step 10.

Step 10: Left branching is possible since $W = \phi$ as shown by vector FIX(J); go to Step 11.

Step 11: Apply the bounding rule and the branching rule.

According to our bounding rule, a free x_{ij} variable is excluded (fixed at 0) for which

$$(c_{ij\ell} - c_{i_1 j\ell}) > (\text{BUBS} - \text{LOWB}) \text{ for } j\varepsilon\bar{W} \text{ and } (i,j)\varepsilon\bar{S} \qquad (51)$$

For x_{13}, (210,381.4375-145,201.5) > (779,502.0-729,839.3125). This also holds for x_{33} and x_{14}, i.e., the bounding rule results in fixing x_{13}, x_{33}, and x_{14} at 0. This is shown in the printout by matrix CX(I,J) where the corresponding variables have been assigned the value 2 because of the bounding rule.

The branching rule directs us to select a free x_{ij} variable corresponding to $c_{i_1 j\ell}$ for which $D_{j\ell} = c_{i_2 j\ell} - c_{i_1 j\ell}$ is maximized over all j. In the printout, $c_{i_2 j\ell}$, $c_{i_1 j\ell}$, and $D_{j\ell}$ are represented by NMINC(J), MINC(J) and DIFBR(J), respectively. Since D_{21} is the maximum, x_{42} is selected as the next left branching variable. This is shown in the printout by BR1 and is represented as (100 i + j) e.g., 402 .

Using the terminology employed in Figures 4a and 4b, the x_{ij} variables fixed at 0 or 1 in the partial solution S_1 will be shown as $S_1 = \{-103, -303, -104, 402\}$. In the computer printout, vector STX displays the x_{ij} variables fixed at 0 or 1. The representation of the variables is, however, somewhat different. An x_{ij} variable fixed at 0, due to any rule, is shown as - (100 i + j) - 1,000,000, e.g., x_{13} is shown as - 1,000,103; an x_{ij} variable fixed at 1 due to the branching rule is represented as (100 i + j), e.g., x_{42} as 402; and an x_{ij} variable fixed at 1 due to a rule other than the branching rule is shown as (100 i + j) + 1,000,000, e.g., x_{23} is represented as 1,000,203.

In the printout, vector STX represents updated partial solution S_1.

We now move to Node 2.

Node 2

The updated matrix CX(I,J) and vector FIX(J) are displayed in the printout.

Step 2: Apply the facility usage rule and the capacity rule for $k=1,2,...,8$.

As shown in the printout, MINSD (representing $\sum_j \bar{d}_{jk\ell}$) , being positive for $\ell=1,2,3,4$, and 5 , these facilities are forced into usage. Further, for $k=4$, expression (47) holds for x_{34} and x_{54} , i.e.,

$$(180-0) > (200-30), \text{ and}$$

$$(180-0) > (200-30), \text{ respectively.}$$

As shown by matrix CX(I,J) in the printout, these two variables are excluded (fixed at 0) by the capacity rule. Since the capacity rule results in fixing at least one variable in the first cycle, another cycle is repeated as displayed in the printout. The second cycle does not fix any more variables. Vector STX is updated accordingly.

Step 3: Solve the relaxed problem (PR_2) .

δ_{k2} represented by FLB(K) , c_{ij2} represented by matrix C(I,J), and solution Y represented by SOLX(J) are displayed in the printout.

Step 4: Compute LOWB.

LOWB, from the printout, is equal to 749011 4375.

Step 5: Compare LOWB with BUBS.

Since LOWB < BUBS, go to Step 6.

Step 6: Check if solution X is feasible in (P).

In the printout, for $k=4$, NSUMD = 290 > 200 , i.e., expression (42) is not satisfied, and we go to Step 10.

Step 10: As shown by vector FIX(J), left branching is possible and we go to Step 11.

Step 11: Apply the bounding rule and the branching rule.

As displayed by matrix CX(I,J) in the printout, the bounding rule results in fixing x_{21} and x_{24} at 0. Now, for $j=4$, except for x_{44}, all the x_{ij} variables are fixed at 0; therefore x_{44}, is fixed at 1. This is reflected by matrix CX(I.J), and vector FIX(J). Vector STX is updated accordingly.

The branching rule selects x_{41} as the next branching variable. This is shown in the printout by BR1, and vector STX is updated accordingly.

We now move to Node 3.

Node 3:

The updated matrix CX(I,J) and vector FIX(J) are displayed in the printout.

Step 2: Apply the facility usage rule and the capacity rule for $k=1,2,\ldots,8$.

The facility usage rule forces facilities 1 to 5, and 8 into usage. For $k=4$, the capacity rule excludes x_{43} and x_{53}, i.e., fixes them at 0; and for $j=3$, all but x_{23} being fixed at 0, x_{23} is fixed at 1. This is displayed in the printout by matrix CX(I,J) and vector FIX(J). Vector STX is updated accordingly.

Although the capacity rule has fixed at least one x_{ij} variable during the initial cycle, another cycle is not necessary, as displayed by vector FIX(J) which represents set W, since we have an x_{ij} variable fixed at 1 for each of the n columns (activities).

Step 3: Solve the relaxed problem (PR_3).

SOLX(J) displays the solution for the relaxed problem.

Step 4: Compute LOWB.

LOWB, shown in the printout, is equal to 779502.0 .

Step 5: Compare LOWB with BUBS.

Since LOWB = BUBS, go to Step 12.

Step 12: Backtrack.

We backtrack by moving to the parent Node 2, and branching to the right by setting $x_{41} = 0$ (since the right branch has not yet been explored). In the printout, this is accomplished by observing the last entry in vector STX, and moving backwards, one entry at a time, until we find a positive entry without 1,000,000 added to it. The corresponding x_{ij} variable is fixed at 0, and we move to the right branch node. Matrix CX(I,J), vector FIX(J) and vector STX are updated accordingly. As displayed in the printout, entry 401 in vector STX is such an entry, and variable x_{41} is fixed at 0 for branching to the right. This is shown in the printout by BRO as 401. The updated vector STX is also displayed.

We now move to Node 4.

<u>Node 4</u>

The updated matrix CX(I,J) and vector FIX(J) are displayed in the printout.

Step 2: Apply the facility usage rule and the capacity rule for $k=1,2,\ldots,8$.

As displayed in the printout, for k=4, MINSD=230 > 200 , i.e., the right-hand side of inequality (47), $(s_k - \sum_j \bar{d}_{jk\ell}) < 0$, and according to our backtracking rules, we backtrack, i.e., go to Step 12.

Step 12: Backtrack.

We backtrack to the parent Node 2, and since the right-hand branch has already been explored, backtrack to Node 1 and to the right-hand branch by fixing x_{42} to 0 . This is shown in the printout by BRO as 402, and vector STX is updated accordingly.

We now move to the next node, i.e., Node 5.

Branch-and-Bound Tree

We continue the branch-and-bound procedure from one node to another until we backtrack to the root node and find that the right branch has already been explored. The procedure, then, terminates and the solution corresponding to the best upper bound is the optimal solution.

For this problem, a total of nine nodes are explored and the optimal value equals 779502.0. The optimal solution is $x_{41} = x_{42} = x_{23} = x_{44} = 1$ and $y_1 = y_2 = y_3 = y_4 = y_5 = y_8 = 1$. This is displayed in the computer printout on the last page of Appendix B.

Figure 7a presents the branch-and-bound tree for this problem, and shows the node numbers, the bounds, and the branching variables.

In order to demonstrate the role of the capacity rule and the bounding rule, Figure 7b displays the x_{ij} variables fixed as 0 or 1 by these rules for this test problem.

The cumulative effect of the various rules, including the facility usage rule, the capacity rule, and the bounding rule, makes the branch-and-bound procedure quite efficient. Further, the storage and updating of the x_{ij} variables fixed at 0 or 1 is done in a manner that makes utmost use of the relevant information at the preceding node.

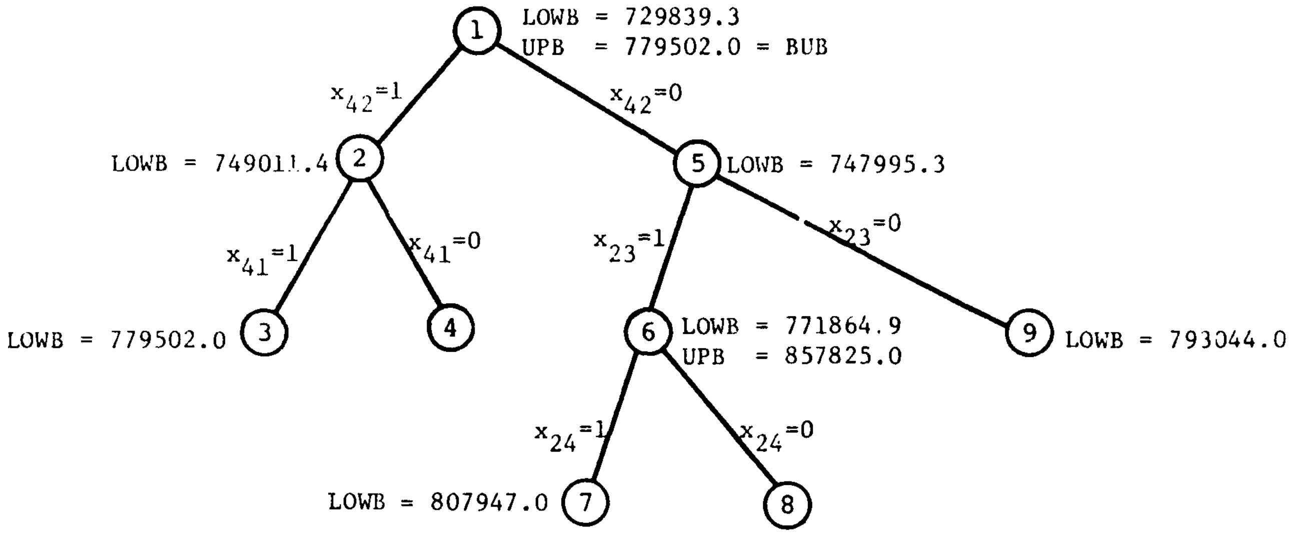


Figure 7a

Branch-and-bound tree for a test problem
(Test Problem with m=5, n=4, and p=8)

Node	Capacity Rule	Bounding Rule
1		$x_{13}=0$, $x_{33}=0$, $x_{14}=0$
2	$x_{34}=0$, $x_{54}=0$	$x_{21}=0$, $x_{24}=0$, $x_{44}=1$
3	$x_{43}=0$, $x_{53}=0$, $x_{23}=1$	
5		$x_{43}=0$, $x_{34}=0$
6		$x_{11}=0$, $x_{21}=0$, $x_{31}=0$, $x_{22}=0$, $x_{54}=0$
8	$x_{44}=1$	
9	$x_{53}=1$, $x_{44}=0$, $x_{54}=0$, $x_{24}=1$	

Figure 7b

Variables fixed by the capacity rule and the bounding rule

5. COMPUTATIONAL TEST RESULTS

The computer program ZIPCAP has been tested on several problems. Although primarily designed for capacitated problems (i.e., where the capacity constraints are active), the program can also be used for solving uncapacitated problems as a special case. Since the data available for capacitated problems were limited, some uncapacitated problems were also considered for testing the program. (Most of the data were furnished by Professor Pinkus and are related to his work on multi-echelon inventory systems.)

Table 4 presents the test results of ZIPCAP. In order to verify the optimal solutions, the test problems were also solved by using the 0-1 integer programming code RIP30C [Geoffrion and Nelson (1968)].

In the table, the problem size shows the number of designs (m), activities (n), and facilities (p). This is equivalent to solving a problem having $mn+p$ variables and $n+p$ constraints. The elapsed time represents the time in seconds to solve the problem, excluding the time to read and write the input and to write the output. The total number of nodes explored by ZIPCAP for a specified set of options is also shown.

Both RIP30C and ZIPCAP were run on an IBM 3031 at The George Washington University. The last problem in the table was not run using RIP30C because of the code's capacity limitation to 90 variables and 50 constraints.

The test problem with $m=3$, $n=4$, and $p=5$ has three variations, using different values for the facility capacities. The data for the variable costs a_{ij} , fixed cost b_k , and the capacity requirements d_{ijk} are given in the other document, i.e., Chhabra and Soland (1980).

For the test problem with $m=5$, $n=4$, and $p=8$, runs 4a, 4b, and 4c are the same except for the different intermediate steps' option

TABLE 4

ZIPCAP TEST RESULTS

Problem Size					Capacitated/ Uncapacitated	Data Information	Run Number	CODE USED								
								RIP30C	ZIPCAP							
								Elapsed Time in Seconds	Options						Elapsed Time in Seconds	Number of Nodes
m	n	p	Number of Variables	Number of Constraints					IINPT	ICAPR	ISTEP	IUNCAP	EPS	ET		
3	4	5	[17	9]	Capacitated											
					s_k = 400,400, 1000,400,400	As given in the other document	1	0.987	1	1	0	0	0.0		0.017	3
					s_k = 700 ∀k	-,,-	2	0.679	1	1	0	0	0.0		0.035	9
					s_k = 3000 ∀k	-,,-	3	1.144	1	1	0	0	0.0		0.018	3
5	4	8	[28	12]	Capacitated	Given in Appendix B	4a	2.421	1	1	0	0	0.0		0.082	9
							4b		1	1	1	0	0.0		0.144	9
							4c		1	1	2	0	0.0		2.033	9
							4d		1	0	0	0	0.0		0.124	19
10	8	8	[38	16]	Uncapacitated	As given in the other document	5	485.8	1	0	0	1	0.0		0.529	23
10	30	8	[308	38]	Uncapacitated		6a		1	0	0	1	0.002	55.0	8.229	125
							6b		1	0	0	1	0.0	55.0	19.397	277

(ISTEP) and this results in slight differences in the time taken to solve the problem. Runs 4a and 4d differ in that 4d does not use the capacity rule; the resulting difference in the total number of nodes explored to reach the optimal value points to the effectiveness of the capacity rule in conjunction with the bounding rule.

Run 5 shows the results for an uncapacitated problem with $m=10$, $n=8$, and $p=8$. Option ICAPR is not used since the capacity rule is not useful for an uncapacitated problem.

Another uncapacitated problem with $m=10$, $n=30$, and $p=8$ is solved in runs 6a and 6b. In run 6a, the epsilon value (EPS) is specified as 0.002. The solution value found by exploring 125 nodes may be suboptimal but is guaranteed to be within +0.2 percent of the optimal solution value. Run 6b is made with an epsilon value (EPS) of 0.0, and the optimal solution value is found in 277 nodes. A comparison of runs 6a and 6b shows that the number of nodes is less than half for a solution value that may be suboptimal but very close to the optimal solution value as compared to the number of nodes for an optimal solution value.

In general, a small difference between a solution value that may be suboptimal and the optimal solution value, translates into a significant difference in the corresponding number of nodes and the solution time required.

6. FURTHER CONSIDERATIONS

It was mentioned in Chapter 2 that it is possible to consider alternative formulations of problem (P), and also to consider choices of Lagrange multipliers other than $v_k = b_k$ with the purpose of obtaining "tighter" bounds which, in turn, would further improve the efficiency of the branch-and-bound procedure. These aspects will be discussed in this Chapter.

6.1 Alternative Formulations

Problem (P) can be reformulated by adding additional constraints such that the corresponding Lagrangian relaxation(s), if solved, would provide "tighter" bounds. If such a relaxation does not possess the integrality property, then it provides an equal or better bound compared to that from an LP relaxation, as mentioned in Chapter 2.

Two alternative formulations of problem (P), along with their Lagrangian relaxations, are given below.

6.1.1 Alternative Formulation 1

Formulation (AP1) is obtained by adding the constraints $e_{ik}\, x_{ij} \le y_k$, for all i,j, and k, to problem (P), i.e.,

$$(\text{AP1}) \left\{ \begin{array}{llll} \text{Minimize} & \sum_i \sum_j a_{ij}\, x_{ij} + \sum_k b_k y_k & & (4) \\ \text{subject to} & \sum_i x_{ij} = 1 & \forall j & (2) \\ & \sum_i \sum_j r_{ijk}\, x_{ij} \le y_k & \forall k & (5') \\ & e_{ik}\, x_{ij} \le y_k & \forall i,j,k & (52) \\ & x_{ij}\, ,\ y_k = 0 \text{ or } 1 & \forall i,j,k & (6) \end{array} \right.$$

Since $e_{ik} = 1$ or 0, each constraint of (52) is either equivalent to $x_{ij} \leq y_k$ (if $e_{ik} = 1$) or else is redundant (if $e_{ik} = 0$). Problem (AP1) thus has, at most, mnp additional constraints relative to problem (P). Two Lagrangian relaxations are now considered for problem (AP1).

The first Lagrangian relaxation is obtained with respect to constraints (5') by introducing nonnegative Lagrange multipliers $v_k \geq 0$ to get

$$\text{Minimize} \quad \sum_i \sum_j a_{ij} x_{ij} + \sum_k b_k y_k - \sum_k v_k \left(y_k - \sum_i \sum_j r_{ijk} x_{ij}\right)$$

subject to (2), (52), and (6), or equivalently,

$$(\text{ALR1}_v) \left\{ \begin{array}{llll} \text{Minimize} & \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k v_k r_{ijk}\right) - \sum_k y_k \left(v_k - b_k\right) & & (53) \\ \text{subject to} & \sum_i x_{ij} = 1 & \forall j & (2) \\ & e_{ik} x_{ij} \leq y_k & \forall i,j,k & (52) \\ & x_{ij}, y_k = 0 \text{ or } 1 & \forall i,j,k & (6) \end{array} \right.$$

Another Lagrangian relaxation of problem (AP1) is obtained with respect to constraints (5') and (52) by introducing nonnegative Lagrange multipliers v_k and λ_{ijk}, respectively, to get

$$\begin{aligned} \text{Minimize} \quad & \sum_i \sum_j a_{ij} x_{ij} + \sum_k b_k y_k \\ & - \sum_k v_k \left(y_k - \sum_i \sum_j r_{ijk} x_{ij}\right) \\ & - \sum_i \sum_j \sum_k \lambda_{ijk} \left(y_k - e_{ik} x_{ij}\right) \end{aligned}$$

subject to (2) and (6), or equivalently,

$$(\mathrm{ALR1}_{v,\lambda})\begin{cases} \text{Minimize} & \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k v_k\, r_{ijk} + \sum_k e_{ik}\, \lambda_{ijk}\right) \\ & - \sum_k y_k \left(v_k + \sum_i \sum_j \lambda_{ijk} - b_k\right) & (54) \\ \text{Subject to} & \sum_i x_{ij} = 1 \qquad \forall j & (2) \\ & x_{ij}\,,\ y_k = 0 \text{ or } 1 \qquad \forall i,j,k & (6) \end{cases}$$

For this problem, the solution is:

$$y_k = 0 \quad \text{if} \quad \left(v_k + \sum_i \sum_j \lambda_{ijk} - b_k\right) \leq 0\ ,$$
$$= 1 \quad \text{if} \quad \left(v_k + \sum_i \sum_j \lambda_{ijk} - b_k\right) \geq 0\ ,$$

and $x_{ij} = 1$ if i minimizes $\left(a_{\underline{\ell}j} + \sum_k v_k\, r_{\underline{\ell}jk} + \sum_k e_{\underline{\ell}k}\, \lambda_{\underline{\ell}jk}\right)$

over $\underline{\ell}$.

We need good choices of Lagrange multipliers v_k with which to solve problem ($\mathrm{ALR1}_v$), and of Lagrange multipliers v_k and λ_{ijk} with which to solve problem ($\mathrm{ALR1}_{v,\lambda}$). Problem ($\mathrm{ALR1}_v$) does not possess the integrality property, thus offering the hope of a tight bound, but has more constraints and is difficult to solve compared to problem ($\mathrm{ALR1}_{v,\lambda}$) which, on the other hand, involves more Lagrange multipliers.

6.1.2 Alternative Formulation 2

Another formulation of problem (P) is similar to problem (AP1) except for a modification in constraints (5'), i.e.,

$$(\text{AP2})\begin{cases}\text{Minimize} & \sum_i \sum_j a_{ij}\, x_{ij} + \sum_k b_k y_k & & (4)\\ \text{Subject to} & \sum_i x_{ij} = 1 & \forall j & (2)\\ & \sum_i \sum_j r_{ijk}\, x_{ij} \leq 1 & \forall k & (55)\\ & e_{ik}\, x_{ij} \leq y_k & \forall i,j,k & (52)\\ & x_{ij}\,,\ y_k = 0 \text{ or } 1 & \forall i,j,k & (6)\end{cases}$$

A Lagrangian relaxation with respect to constraints (55) and (52) is obtained by introducing nonnegative Lagrange multipliers v_k and λ_{ijk} to get

$$\begin{aligned}\text{Minimize}\quad & \sum_i \sum_j a_{ij}\, x_{ij} + \sum_k b_k y_k\\ & - \sum_k v_k \left(1 - \sum_i \sum_j r_{ijk}\, x_{ij}\right)\\ & - \sum_i \sum_j \sum_k \lambda_{ijk} \left(y_k - e_{ik}\, x_{ij}\right)\end{aligned}$$

Subject to (2) and (6), or equivalently,

$$(\text{ALR2}_{v,\lambda})\begin{cases}\text{Minimize} & \sum_i \sum_j x_{ij} \left(a_{ij} + \sum_k v_k\, r_{ijk} + \sum_k e_{ik}\, \lambda_{ijk}\right) & & \\ & + \sum_k y_k \left(b_k - \sum_i \sum_j \lambda_{ijk}\right) - \sum_k v_k & & (56)\\ \text{Subject to} & \sum_i x_{ij} = 1 & \forall j & (2)\\ & x_{ij}\,,\ y_k = 0 \text{ or } 1 & \forall i,j,k & (6)\end{cases}$$

For this problem, the solution is:

$$\begin{aligned} y_k &= 0 \quad \text{if} \quad \sum_i \sum_j \lambda_{ijk} \leq b_k\,,\\ &= 1 \quad \text{if} \quad \sum_i \sum_j \lambda_{ijk} \geq b_k\,,\end{aligned}$$

and $x_{ij} = 1$ if i minimizes $\left(a_{\ell j} + \sum_k v_k r_{\ell jk} + \sum_k e_{\ell k} \lambda_{\ell jk}\right)$ over ℓ .

Here again, we need good choices of the Lagrange multipliers v_k and λ_{ijk} with which to solve problem (ALR2$_{v,\lambda}$).

6.1.3 Choice of Lagrange Multipliers

Each of the relaxations (ALR1$_{v,\lambda}$) and (ALR2$_{v,\lambda}$) involves p v_k Lagrange multipliers and mnp λ_{ijk} multipliers. If we have good choices of these multipliers, the resulting solutions of the relaxed problems should provide "tighter" bounds (because of the additional constraints) than the bound from relaxation (LR$_v$). Since relaxations (ALR1$_{v,\lambda}$) and (ALR2$_{v,\lambda}$) are similar to a great extent, only the relaxation (ALR1$_{v,\lambda}$) will be considered for further discussion.

By looking at expression (54) of the formulation (ALR1$_{v,\lambda}$), a meaningful choice of the Lagrange multipliers v_k and λ_{ijk} appears to follow from setting

$$v_k + \sum_i \sum_j \lambda_{ijk} = b_k \qquad \forall k \tag{57}$$

so that each of the λ_{ijk} can be chosen as

$$\left.\begin{aligned} \lambda_{ijk} &= \frac{b_k - v_k}{n\left(\sum_i e_{ik}\right)} \qquad \text{if } e_{ik} = 1 , \\ &= 0 \text{ otherwise} \end{aligned}\right\} \tag{58}$$

The solution for problem (ALR1$_{v,\lambda}$) is then to select, from each column j , an x_{ij} variable which minimizes $\left(a_{\ell j} + \sum_k v_k r_{\ell jk} + \sum_k e_{\ell k} \lambda_{\ell jk}\right)$ over ℓ .

Arbitrary values were considered for the v_k (e.g., v_k equal to $3/4\ b_k$, $1/2\ b_k$, $1/4\ b_k$, and 0) , the λ_{ijk} were then computed from

(58), and the test problem with three designs (m), four activities (n) and five facilities (p) was solved. Three cases with different capacities s_k (as specified in Chapter 5, Table 4) were tried for the solutions at the initial node. The results, however, were not conclusive in terms of providing a meaningful choice of the Lagrange multipliers v_k (and of the λ_{ijk}) .

Since the relaxation ($ALR1_{v,\lambda}$) possesses the integrality property, a choice of the multipliers as the optimal values of the dual variables of the corresponding linear programming problem would provide a solution as good as the LP solution (as stated in Chapter 2). We do not propose to solve linear programs as a part of our branch-and-bound methodology. However, we have made some LP runs, basically to see if the results provide insight leading to the choice of the Lagrange multipliers, and also to see if the resulting LP solutions are "close" to the integer solutions. These results are given below.

The LP formulation ($\overline{AP1}$) corresponding to problem (AP1) is:

$$(\overline{AP1})\left\{\begin{array}{llll}
\text{Minimize} & \sum_i \sum_j a_{ij} x_{ij} + \sum_k b_k y_k & & (4) \\
\text{Subject to} & \sum_i x_{ij} = 1 & \forall j & (2) \\
 & \sum \sum r_{ijk} x_{ij} \leq y_k & \forall k & (5') \\
 & e_{ik} \; x_{ij} \leq y_k & \forall i,j,k & (52) \\
 & y_k \leq 1 & \forall k & (15) \\
 & x_{ij} \, , \, y_k \geq 0 & \forall i,j,k & (16)
\end{array}\right.$$

The constraints $x_{ij} \leq 1$ are implicit in constraints (2).

Problem ($\overline{AP1}$) was solved for the test problem with $m=3$, $n=4$, and $p=5$ and three different cases for the capacities s_k (as specified in

Chapter 5, Table 4). Each case was solved using the IMSL (International Mathematical and Statistical Library) Code ZX3LP on an IBM 3031 at The George Washington University.

Note that the formulation $(\overline{AP1})$ has up to mnp more constraints than the LP formulation $(\bar{P})$ given in Chapter 2. For our test problem, this translates into solving a problem of 17 variables and 50 constraints corresponding to formulation $(\overline{AP1})$ as against 17 variables and 14 constraints corresponding to formulation $(\bar{P})$.

Table 5 lists the solution values for each of the three cases with different capacities for the small problem with three designs, four activities and five facilities. The solutions to problems $(\bar{P})$ and $(\overline{AP1})$, obtained from ZX3LP, show the optimal solution values, the optimal values of the variables x_{ij} and y_k, and the optimal values of the dual variables corresponding to the Lagrangian relaxations (LR_v) and $(ALR1_{v,\lambda})$, i.e., v_k associated with the capacity constraints (3') and λ_{ijk} associated with the constraints (52). The table also shows Z(P), and the Lagrangian solution value $Z(LR_v)$ obtained by setting $v_k = b_k$ for all k at the root node, i.e., $Z(LR_b)$.

As expected, the LP solutions for each of the three cases show $Z(\overline{AP1})$ to be considerably higher than $Z(\bar{P})$, and closer to Z(P), thereby providing a tighter bound. As for the Lagrange multipliers v_k and λ_{ijk}, the following relationships are observed.

$$\sum_i \sum_j \lambda_{ijk} \leq b_k \qquad \forall k \text{, and}$$

$$v_k + \sum_i \sum_j \lambda_{ijk} \geq b_k \qquad \forall k \text{ .}$$

Also, for $v_k = 0$, $\sum_i \sum_j \lambda_{ijk} = b_k$, and

for $v_k \geq b_k$, $\sum_i \sum_j \lambda_{ijk} = 0 \quad \forall k$.

TABLE 5

LP AND OTHER SOLUTION VALUES FOR A TEST PROBLEM
(m=3, n=4, AND p=5)

Test Problem Case	Z(P)	$Z(\bar{P})$	$Z(LR_b)$	$Z(\overline{API})$
s_k = 700 ∀k	37,774.0 $x_{21}=1$ $x_{22}=1$ $x_{23}=1$ $x_{24}=1$ $y_1=1$ $y_3=1$ $y_5=1$	36,688.04 $x_{11}=0.2$ $x_{32}=1$ $x_{23}=1$ $x_{24}=1$ $x_{21}=0.8$ $y_1=0.85$ $y_2=0.05$ $y_3=1$ $y_4=0.05$ $y_5=0.85$ $v_1=1750$ $v_2=2000$ $v_3=2221$ $v_4=1350$ $v_5=1000$	36,504.92 $x_{11}=1$ $x_{32}=1$ $x_{23}=1$ $x_{24}=1$ Solution infeasible in problem (P)	37,678.14 $x_{11}=0.05$ $x_{22}=0.95$ $x_{23}=0.95$ $x_{24}=0.95$ $x_{21}=0.95$ $x_{32}=0.05$ $x_{33}=0.05$ $x_{34}=0.05$ $y_1=0.95$ $y_2=0.05$ $y_3=0.95$ $y_4=0.05$ $y_5=0.95$ $v_1=0$ $v_2=0$ $v_3=1491.2$ $v_4=0$ $v_5=0$ $\lambda_{211}=377.2$ $\lambda_{332}=1621.9$ $\lambda_{233}=258.8$ $\lambda_{324}=838.9$ $\lambda_{225}=1000.0$ $\lambda_{231}=366.8$ $\lambda_{342}=378.1$ $\lambda_{344}=511.1$ $\lambda_{241}=1006.0$
s_k=400,400 1000,400, 400	40,174.0 $x_{11}=1$ $x_{32}=1$ $x_{33}=1$ $x_{24}=1$ $y_1=1$ $y_2=1$ $y_3=1$ $y_4=1$ $y_5=1$	37,472.62 $x_{11}=1$ $x_{32}=1$ $x_{23}=0.8$ $x_{14}=0.2$ $x_{33}=0.2$ $x_{24}=0.8$ $y_1=1$ $y_2=0.2$ $y_3=1$ $y_4=0.2$ $y_5=1$ $v_1=1750$ $v_2=2000$ $v_3=5144$ $v_4=1350$ $v_5=3879$	36,775.5 $x_{11}=1$ $x_{32}=1$ $x_{13}=1$ $x_{24}=1$ Solution infeasible in problem (P)	38,360.26 $x_{11}=1$ $x_{22}=0.74$ $x_{23}=0.74$ $x_{14}=0.19$ $x_{32}=0.26$ $x_{33}=0.26$ $x_{24}=0.55$ $x_{34}=0.26$ $y_1=1$ $y_2=0.26$ $y_3=1$ $y_4=0.26$ $y_5=1$ $v_1=4575.0$ $v_2=0$ $v_3=5063.0$ $v_4=0$ $v_5=$[illegible] $\lambda_{322}=553.0$ $\lambda_{324}=657.0$ $\lambda_{332}=1447.0$ $\lambda_{344}=693.0$
s_k=3000 ∀k	37,429.0 $x_{11}=1$ $x_{22}=1$ $x_{23}=1$ $x_{24}=1$ $y_1=1$ $y_3=1$ $y_5=1$	33,156.2 $x_{11}=1$ $x_{32}=1$ $x_{23}=1$ $x_{24}=1$ $y_1=0.15$ $y_2=0.01$ $y_3=0.32$ $y_4=0.01$ $y_5=0.15$ $v_1=1750$ $v_2=2000$ $v_3=1750$ $v_4=1350$ $v_5=1000$	33,156.2 $x_{11}=1$ $x_{32}=1$ $x_{23}=1$ $x_{24}=1$	36,754.0 $x_{11}=0.5$ $x_{22}=0.5$ $x_{13}=0.5$ $x_{24}=0.5$ $x_{21}=0.5$ $x_{32}=0.5$ $x_{23}=0.5$ $x_{34}=0.5$ $y_1=0.5$ $y_2=0.5$ $y_3=0.5$ $y_4=0.5$ $y_5=0.5$ $v_1=0$ $v_2=0$ $v_3=0$ $v_4=0$ $\lambda_{221}=915.0$ $\lambda_{322}=1412.0$ $\lambda_{113}=1020.0$ $\lambda_{324}=1350.0$ $\lambda_{231}=28.0$ $\lambda_{342}=588.0$ $\lambda_{213}=675.0$ $\lambda_{241}=807.0$ $\lambda_{223}=55.0$ $v_5=0$ $\lambda_{225}=1000.0$

Although the relationship among various λ_{ijk} values is not apparent, the above observations are useful in further exploring some good choices of the Lagrange multipliers for the relaxation $(ALR1_{v,\lambda})$, as discussed earlier.

As for the "closeness" of the LP solution to that of the integer solution, most of the solution values x_{ij} and y_k of problem $(\overline{AP1})$ are fractional, and their rounding off to 0 or 1 does not, in general, seem to correspond to the optimal integer solution values x_{ij} and y_k of problem (P).

Table 5 also displays $Z(LR_b)$ at the root node for each of the three cases. For $s_k = 3000$, $Z(LR_b) = Z(\bar{P})$, and the Lagrange multipliers, as reflected by the values of the dual variables of problem $(\bar{P})$, are equal to b_k for all k . This is expected from Theorem 3 and our discussion of the integrality property in Chapter 2. Further, the dual variables of $(\bar{P})$ for the first two cases (i.e., $s_k = 700 \ \forall k$, and $s_k = 400, \ldots$) have values $v_k \geq b_k$ from Theorem 1.

The $Z(LR_b)$ values in Table 5, however, take no consideration of the capacity rule and/or the facility usage rule of our branch-and-bound procedure. These rules, by fixing certain x_{ij} values to 0 or 1, and by forcing certain facilities into the solution, provide an improved lower bound. As per our branch-and-bound procedure the improved lower bound at the root node is obtained by solving problem (PR_1). For example, for $s_k = 700 \ \forall k$, the values of $Z(LR_b)$ and $Z(PR_1)$ are shown in Figure 8. The figure also shows the values of $Z(\bar{P})$, $Z(\overline{AP1})$, and $Z(P)$. The branch-and-bound procedure rules provide an improved value of the lower bound $Z(PR_1)$ compared to $Z(\bar{P})$. It appears that some good values of the Lagrange multipliers of the relaxation $(ALR1_{v,\lambda})$, if found, could, in conjunction with these rules, provide significant improvement over $Z(\overline{AP1})$, and without the need to solve an LP problem.

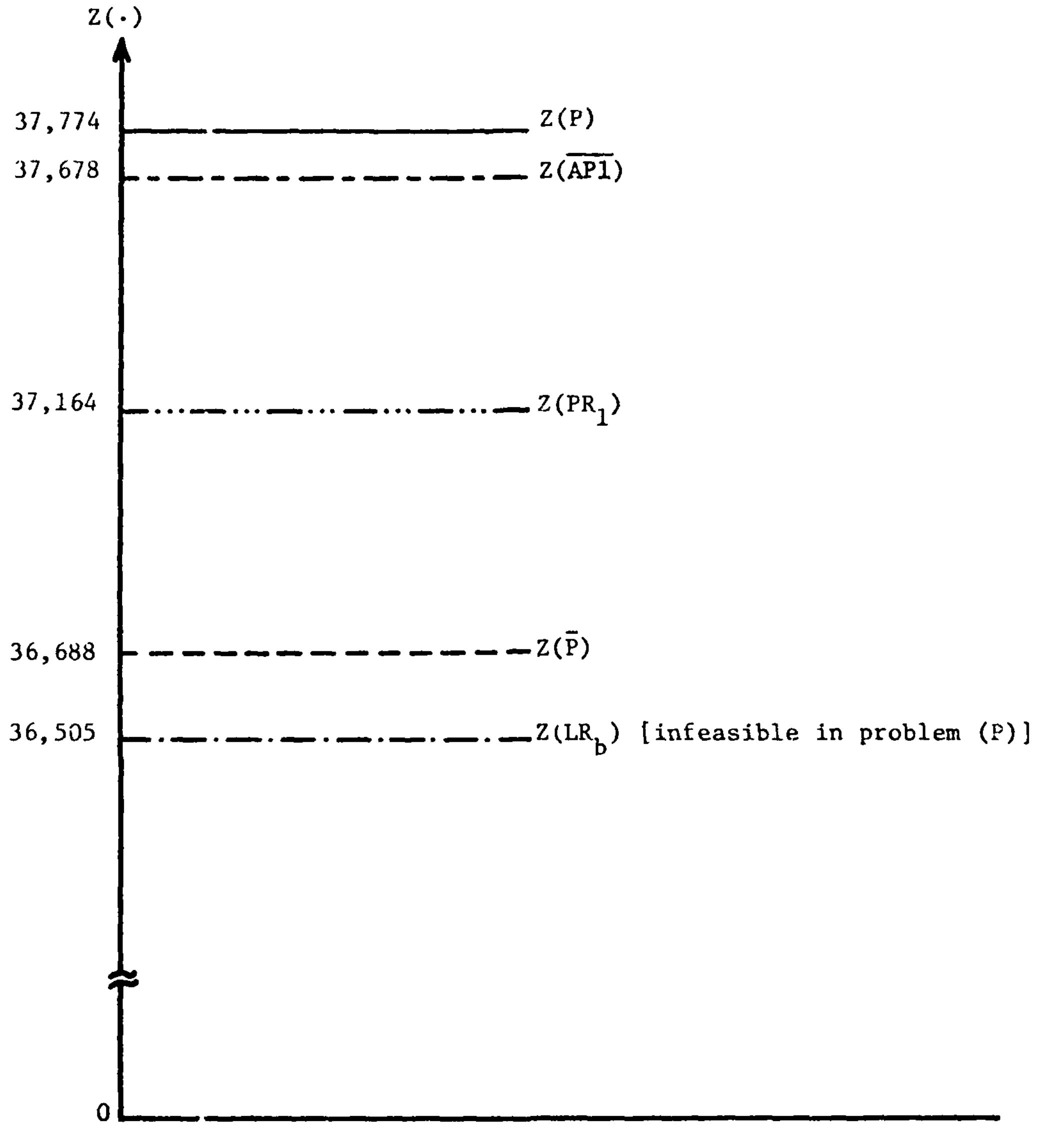


Figure 8

Lagrangian and other solution values for a test problem
(Test problem with m=3, n=4, p=5, and s_k = 700 ∀k)

6.2 Subgradient Method

It was mentioned in Chapter 2 that setting the Lagrange multipliers v_k equal to b_k for all k provides a good starting point in solving the Lagrangian relaxation (LR_v) of problem (P). From Theorem 3, this choice is optimal (in terms of providing the tightest lower bound) if the resulting solution is feasible in problem (P). In other cases, i.e., where the resulting solution is not feasible in problem (P), this choice is generally not optimal and it is possible to tighten the bounds by considering values of $v_k \geq b_k$ (from Theorem 1). One method that seems useful in providing such a choice is the subgradient method.

The subgradient method is an adaptation of the gradient method in which gradients are replaced by subgradients. Through a heuristic choice of the step-size, this method has been successfully used to provide improved bounds and sometimes optimal solutions [for details see Held, Wolfe, and Crowder (1974), Fisher (1978), and Christofides (1980)]. The fundamental theoretical result is that

$$Z(LR_v^g) \rightarrow Z(D) \quad \text{if} \quad t^g \rightarrow 0 \quad \text{and} \quad \sum_{q=0}^{g} t^q \rightarrow \infty \quad \text{as} \quad g \rightarrow \infty ,$$

where t^g is the positive step-size t for the gth iteration, and $Z(LR_v^g)$ is the solution value of the relaxed problem (LR_v) using v_k values obtained at the gth iteration.

In the case of problem (P), the step-size t^{g+1} for iteration $g + 1$, given that we have a solution of (LR_v^g), is given by

$$t^{g+1} = \frac{\lambda^{g+1}[Z^* - Z(LR_v^g)]}{\sum_k ||y_k^g - \sum_i \sum_j r_{ijk} x_{ij}^g||^2} , \tag{59}$$

where λ^{g+1} is a scalar and generally between 0 and 2, and Z^* is an upper bound on $Z(LR_v^g)$, frequently obtained by applying a heuristic to solve problem (P).

Given the vector of multipliers v^g , v^{g+1} is generated by

$$v_k^{g+1} = v_k^g - t^{g+1}\left(y_k^g - \sum_i \sum_j r_{ijk}\, x_{ij}^g\right), \tag{60}$$

where we enforce $v_k^{g+1} \geq b_k$ in our case of problem (P) (because of Theorem 1).

Justification for these rules and computational results of applications of the subgradient method are given in Held et al (1974). The scalar λ is generally initiated by setting $\lambda^1 = 2$ and halving subsequent λ's whenever the resulting solution value has failed to increase in some fixed number of iterations. This rule has performed well empirically [Held et al (1974) and Fisher (1978)].

For the test problem with three designs, four activities, five facilities, and the capacities $s_1 = 400$, $s_2 = 400$, $s_3 = 1000$, $s_4 = 400$, $s_5 = 400$, the Lagrangian solution obtained at the root node by setting $v_k = b_k$ for all k , i.e., the solution to problem (PR_1) is infeasible in problem (P), i.e., it violates the capacity constraints. It seems that the subgradient method could be useful in considering $v_k \geq b_k$ with the ultimate purpose of obtaining a tighter lower bound, and, depending on the revised solution(s), possible improvement in the best upper bound. Another possiblity could be to first arbitrarily increase the relevant v_k values by a small percentage of the b_k values and then solve problem (LR_v), hopefully to improve the lower bound; and thereafter to use the subgradient method for obtaining subsequent values of v_k , and tightening the bounds.

Both of the areas discussed above, i.e., the consideration of alternative formulations of problem (P), and the application of the subgradient method, and their combination, seem useful for continued research in terms of further improving the branch-and-bound procedure for solving the multiactivity multifacility capacity-constrained 0-1 assignment problems.

REFERENCES

BALAS, E. and N. CHRISTOFIDES (1976). Talk presented at the Ninth International Symposium of Mathematical Programming, Budapest.

BAZARAA, M. S. and J. J. GOODE (1977). The traveling salesman problem: A duality approach. Mathematical Programming **13** 221-237.

BURDET, C. A. and E. L. JOHNSON (1975). A subadditive approach to solve linear integer programs. Presented at workshop on integer programming, Bonn.

CHALMET, L. G. and L. F. GELDERS (1976). Lagrangian relaxations for a generalized assignment-type problem. Katholieke Universiteit Leuven Working Paper, No. 76-12.

CHHABRA, K. L. and R. M. SOLAND (1980). Program description and user's guide for ZIPCAP -- a zero-one integer program to solve multi-activity multifacility capacity-constrained assignment problems. Technical Paper Serial T-423, Program in Logistics, The George Washington University.

CHRISTOFIDES, N. (1980). Lagrangian relaxation. Talk presented on February 27, 1980 at the National Bureau of Standards, Maryland.

CORNUEJOLS, G., M. L. FISHER, and G. L. NEMHAUSER (1977). Location of bank accounts to optimize float: An analytical study of exact and approximate algorithms. Management Science **23** 789-810.

ETCHEBERRY, J. (1977). The set-covering problem: A new implicit enumeration algorithm. Operations Research **25** 760-772.

ETCHEBERRY, J., C. CONCA, and E. STACCHETTI (1978). An implicit enumeration approach for integer programming using subgradient optimization. Publication No. 78/04/C, Universidad de Chile.

EVERETT, H. (1963). Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. Operations Research **11** 399-417.

FISHER, M. L. (1973). Optimal solution of scheduling problems using Lagrange multipliers: Part I. Operations Research **21** 1114-1127.

FISHER, M. L. (1976). A dual algorithm for the one-machine scheduling problem. Mathematical Programming **11** 229-251.

FISHER, M. L. (1978). Lagrangian relaxation methods for combinatorial optimization. Decision Sciences Working Paper No. 78-10-06, University of Pennsylvania.

FISHER, M. L. and D. S. HOCHBAUM (1978). Database location in computer networks. Decision Sciences Working Paper No. 78-03-06, University of Pennsylvania.

FISHER, M. L. and L. SCHRAGE (1972). Using Lagrange multipliers to schedule elective hospital admissions. Working Paper, University of Chicago.

FISHER, M. L. and J. F. SHAPIRO (1974). Constructive duality in integer programming. *SIAM Journal on Applied Mathematics* **27** 31-52.

GAVISH, B. (1978). On obtaining the 'best' multipliers for a Lagrangian relaxation for integer programming. *Computers and Operations Research* **5** 55-71.

GEOFFRION, A. M. (1967). Integer programming by implicit enumeration and Bala's method. *SIAM Review* **9** 178-190.

GEOFFRION, A. M. (1974). Lagrangian relaxation for integer programming. *Mathematical Programming Study* **2** 82-114.

GEOFFRION, A. M. (1975). A guide to computer-assisted methods for distribution system planning. *Sloan Management Review* **16** 17-24.

GEOFFRION, A. M. and R. E. MARSTEN (1972). Integer programming algorithms: A framework and state-of-the-art survey. *Management Science* **18** 465-491.

GEOFFRION, A. M. and R. MCBRIDE (1978). Lagrangian relaxation applied to capacitated facility location problems. *AIIE Transactions* **10** 40-47.

GEOFFRION, A. M. and A. B. NELSON (1968). User's instructions for 0-1 integer linear programming code RIP30C. Memorandum RM-5627-PR. The Rand Corporation.

GILMORE, P. C. and R. E. GOMORY (1963). A linear programming approach to the cutting-stock problem, Part II. *Operations Research* **11** 863-888.

GROSS, D. and C. E. PINKUS (1979). Designing a support system for repairable items. *Computers and Operations Research* **6** 59-68.

GROSS, D., C. E. PINKUS, and R. M. SOLAND (1979). Designing a multi-product multi-echelon inventory system. Technical Paper Serial T-392, Program in Logistics, The George Washington University.

HELD, M. and R. M. KARP (1970). The traveling salesman problem and minimum spanning trees. *Operations Research* **18** 1138-1162.

HELD, M. and R. M. KARP (1971). The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming* **1** 6-25.

HELD, M., P. WOLFE, and H. D. CROWDER (1974). Validation of subgradient optimization. *Mathematical Programming* **6** 62-88.

HILLIER, F. S. and G. J. LIEBERMAN (1980). *Operations Research*. Holden-Day, San Francisco.

KHUMAWALA, B. M. (1973). An efficient heuristic procedure for the uncapacitated warehouse location problem. *Naval Research Logistics Quarterly* **20** 109-121.

KHUMAWALA, B. M. and J. P. STINSON (1980). Unpublished Paper.

KNUTH, D. E. (1968). *The Art of Computer Programming*. Addison-Wesley Publishing Company, Reading, Mass.

LORIE, J. and L. I. SAVAGE (1955). Three problems in capital rationing. *Journal of Business* 229-239.

MUCKSTADT, J. and S. A. KOENIG (1977). An application of Lagrangian relaxation to scheduling in power generation systems. *Operations Research* **25** 387-403.

NEMHAUSER, G. L. and G. WEBER (1978). Optimal set partitioning, matchings and Lagrangian duality. Talk delivered at the New York ORSA/TIMS meeting.

PINKUS, C. E. (1971). The design of multi-product multi-echelon inventory systems using a branch-and-bound algorithm. Technical Paper Serial T-250, The Institute for Management Science and Engineering, The George Washington University.

PINKUS, C. E. (1975). Optimal design of multi-product multi-echelon inventory systems. *Decision Sciences* **6** 492-507.

PINKUS, C. E., D. GROSS, and R. M. SOLAND (1973). Optimal design of multiactivity multifacility systems by branch-and-bound. *Operations Research* **21** 270-283.

ROSS, G. and R. M. SOLAND (1975). A branch-and-bound algorithm for the generalized assignment problem. *Mathematical Programming* **3** 91-103.

ROSS, G. and R. M. SOLAND (1977). Modeling facility location problems as generalized assignment problems. *Management Science* **24** 345-357.

ROSS, G. and R. M. SOLAND (1980). A multicriteria approach to the location of public facilities. *European Journal of Operational Research* **4** 307-321.

SHAPIRO, J. F. (1977). A survey of Lagrangian techniques for discrete optimization. Technical Report No. 133, OR Center, MIT.

APPENDIX A

ZIPCAP LISTING (REVISED)

```
FORTRAN IV G LEVEL  21                  MAIN              DATE = 80315         11/07/24
C                       ZIPCAP, A ZERO-ONE INTEGER PROGRAM IS DESIGNED
C                       TO SOLVE MULTIACTIVITY MULTIFACILITY CAPACITY-
C                       CONSTRAINED PROBLEMS HAVING VARIABLE AND FIXED
C                       COSTS. IT ALSO SOLVES UNCAPACITATED PROBLEMS AS A
C                       SPECIAL CASE
      INTEGER   D(35,35,30), A(35,35), CX(35,35), E(35,30),
     1          B(30), BSOLX(35), BSOLY(30), FLB(30),FIX(35),FIXI(35),
     2          FUB(30), S(30), SOLX(35), STX(1225)
      REAL               MINC(35), NMINC(35)
      DIMENSION C(35,35), DIFBR(35), KT2(35), MIND(35)
      INTEGER   BRO, BR1, FC, FCUB, P
      REAL      LOWB, MAXDIF, MINSC
C     **************OPTIONS AVAILABLE: IINPT, ICAPR, ISTEP, IUNCAP,EPS
C                       IINPT=1 IF INPUT LISTING DESIRED; O OTHERWISE
C                       ICAPR=1 IF CAPACITY RULE TO BE USED; O OTHERWISE
C                       ISTEP=0 IF LISTING OF INTERMEDIATE STEPS
C                       NOT DESIRED. ISTEP=1 IF SUMMARY OF BRANCH &
C                       BOUND NODES DESIRED. ISTEP=2 IF DETAILED
C                       LISTING OF INTERMEDIATE STEPS DESIRED.
C                       IUNCAP=1 IF SOLVING AN UNCAPACITATED PROBLEM,
C                       0 OTHERWISE.
C                       EPS= A FRACTIONAL VALUE IF SUBOPTIMAL
C                       SOLUTION DESIRED, E.G., EPSILON AS 0.005
C                       IMPLIES SOLUTION TO BE WITHIN +0.5 PERCENT
C                       OF THE OPTIMAL SOLUTION. EPS=0.0 IF OPTIMAL
C                       SOLUTION DESIRED.
C                       ET= ELAPSED TIME IN SECONDS, IF SPECIFIED, AT
C                       WHICH THE NODE AND BOUNDS RELATED INFORMATION
C                       IS PRINTED. THIS IS USEFUL IN A SITUATION IF
C                       ISTEP=0 AND THE PROGRAM TERMINATES BEFORE
C                       REACHING THE FINAL SOLUTION.
C     **************READ INPUT DATA***********************************
      READ 10, IINPT, ICAPR, ISTEP, IUNCAP, EPS, ET
   10 FORMAT (4I1, F6.5, F10.3)
C                       M= NUMBER OF DESIGNS
C                       N= NUMBER OF ACTIVITIES
C                       P= NUMBER OF FACILITIES
      READ 20,M,N,P
   20 FORMAT (3I5)
C                       A(I,J): VARIABLE COST MATRIX
      READ 30,  ((A(I,J), I=1,M),J=1,N)
   30 FORMAT   (8I10)
C                       B(K): FIXED COST VECTOR
      READ 30, (B(K),K=1,P)
      IF (IUNCAP.EQ.1) GO TO 40
C                       S(K): CAPACITY LIMIT VECTOR; REQUIRED ONLY
C                       IF IUNCAP=0
      READ 30, (S(K),K=1,P)
C                       D(I,J,K): CAPACITY USAGE MATRIX; REQUIRED
C                       ONLY IF IUNCAP=0
      DO 32 K=1,P
      READ 30,((D(I,J,K), I=1,M),J=1,N)
   32 CONTINUE
      DO 37 K=1,P
      DO 37 I=1,M
         IF (D(I,1,K).EQ.0) GO TO 35
         E(I,K)=1
         GO TO 37
```

AD-A102 583 GEORGE WASHINGTON UNIV WASHINGTON DC PROGRAM IN LOGISTICS F/G 12/1
SOLVING MULTIACTIVITY MULTIFACILITY CAPACITY-CONSTRAINED 0-1 AS--ETC(U)
MAY 81 K L CHHABRA N00014-80-C-0169
UNCLASSIFIED SERIAL-T-441 NL

2 OF 2
AD A
102583

END
DATE
FILMED
9-81
DTIC

```
   35    E(I,K)=0
   37 CONTINUE
      GO TO 90
C                    E(I,K): DESIGN-FACILITY MATRIX; REQUIRED ONLY
C                    IF IUNCAP=1
   40 READ 45,((E(I,K),I=1,M),K=1,P)
   45 FORMAT  (80I1)
      DO 80 K=1,P
         S(K)=N
      DO 75 I=1,M
         IF (E(I,K).EQ.1) GO TO 65
      DO 60 J=1,N
         D(I,J,K)=0
   60 CONTINUE
         GO TO 75
   65 DO 70 J=1,N
         D(I,J,K)=1
   70 CONTINUE
   75 CONTINUE
   80 CONTINUE
C     **************PRINT INPUT DATA*********************************
   90 PRINT 95, IINPT,  ICAPR, ISTEP, IUNCAP, EPS, ET
   95 FORMAT ( '1',' OPTIONS SELECTED :  IINPT=',I1,
     1        '  ICAPR=',I1, '  ISTEP=',I1, '  IUNCAP=',I1,
     2        '  EPS=',F8.5, '  ET=', F10.3///)
      IF (IINPT.EQ.0) GO TO 168
      PRINT  100,M,N,P
  100 FORMAT ('0', T55, 'INPUT DATA',/1X, T55, '----- -----',//////1X,
     1T41, 'NUMBER OF DESIGNS (M)=', 4X,I4//1X,T41,
     2'NUMBER OF ACTIVITIES (N)=', 1X,I4//1X, T41,
     3'NUMBER OF FACILITIES (P)=',1X, I4///)
      PRINT  105
  105 FORMAT ( 4X,    'VARIABLE COST MATRIX  A(I,J)',/4X,
     1'-----------------------------',/)
      DO  110 I=1,M
  110 PRINT 115, I, (A(I,J),J=1,N)
  115 FORMAT ('0', T6, 'I=', I3, 4X,8I13, 4(/, 14X,8I13))
      PRINT 120
  120 FORMAT('0',//4X,'FIXED COST VECTOR  B(K)',/4X,
     1'-----------------------',      /)
      PRINT 122, (B(K),K=1,P)
  122 FORMAT ('0', T15, 8I13, 3(/, 14X,8I13))
      PRINT  125
  125 FORMAT('0',//4X,'CAPACITY LIMIT VECTOR  S(K)',/4X,
     1'---------------------------',  /)
      PRINT 128, (S(K),K=1,P)
  128 FORMAT ('0', T15, 8I13, 3(/, 14X,8I13))
      PRINT 130
  130 FORMAT('0',//4X, 'CAPACITY USAGE MATRIX  D(I,J,K)',/4X,
     1'-------------------------------',/)
      DO  150 K=1,P
      PRINT  135,K
  135 FORMAT ('0',//5X,'K=',I3/)
      DO  145 I=1,M
      PRINT  140,I,(D(I,J,K), J=1,N)
  140 FORMAT ('0', T6, 'I=', I3, 4X,8I13, 4(/, 14X,8I13))
  145 CONTINUE
  150 CONTINUE
```

```
FORTRAN IV G LEVEL  21                MAIN              DATE = 80315          11/07/24

            PRINT 155
        155 FORMAT('0',//4X,'DESIGN-FACILITY MATRIX E(I,K)',/4X,
          1'------------------------------',/)
            DO 160  I=1,M
            PRINT 158, I, (E(I,K),K=1,P)
        158 FORMAT ('0', T6, 'I=', I3, 4X,8I13, 3(/, 14X,8I13))
        160 CONTINUE
        168 IF (ISTEP.EQ.0) GO TO 190
            IF (ISTEP.EQ.1) GO TO 175
            PRINT 170
        170 FORMAT ('0',///55X,'DETAILED LISTING OF STEPS',/)
            GO TO 190
        175 PRINT 180
        180 FORMAT ('0',///55X,'SUMMARY OF STEPS',/)
      C     *************INITIALIZE**************************************
        190 BUB=9999999.
            BUBS= BUB/ (1.0+EPS)
            NSX=0
            NOD=1
            IBNOD=1
            INET=0
            INSET=0
            DO 205 J=1,N
               FIX(J)=0
               KT2(J)=0
            DO 205 I=1,M
               CX(I,J)=0
        205 CONTINUE
            LQ1=0
            LQ2=0
            LR2=0
            CALL TIMET(ITO)
            IF (ISTEP.EQ.0) GO TO 208
            PRINT 220,NOD
        208 IF(NSX.EQ.0) GO TO 283
      C                    CX(I,J) CONTAINS FIXED AND FREE X(I,J) VARIABLES.
      C                     STX(INS) CONTAINS FIXED X(I,J) VARIABLES.
      C                     CX(I,J) AND STX(INS) ARE UPDATED BY THE CAPACITY
      C                     RULE, THE BOUNDING RULE, AND THE RULE FOR
      C                     BRANCHING AND BACKTRACKING.
      C                     IN CX(I,J) A FIXED VARIABLE IS RECORDED AS 1 OR
      C                     2, AND A FREE VARIABLE AS 0.
      C                     A VALUE OF 1 IMPLIES THAT THAT PARTICULAR VARIABLE
      C                     IS FIXED, AND FIX(J) IS SET EQUAL TO 1 IMPLYING
      C                     THAT COLUMN J HAS A FIXED VARIABLE OF VALUE 1.
      C                     A VALUE OF 2 IMPLIES THAT THAT PARTICULAR VARIABLE
      C                     SHOULD NOT BE CONSIDERED FOR CURRENT COMPUTATIONS.
      C                     AN X(I,J) RECORDED IN CX(I,J) AS 1 DUE TO THE
      C                     BRANCHING RULE IS RECORDED IN STX(INS) AS X*100+J.
      C                     AN X(I,J) RECORDED IN CX(I,J) AS 1 DUE TO THE
      C                     CAPACITY RULE OR THE BOUNDING RULE IS RECORDED IN
      C                     STX(INS) AS (X*100+J)+1000000.
      C                     AN X(I,J) RECORDED IN CX(I,J) AS 2 IS RECORDED IN
      C                     STX(INS) AS -(X*100+J)-1000000.
        210 IF (ISTEP.EQ.0) GO TO 225
        215 PRINT 220,NOD
        220 FORMAT ('0',//6X,'NODE NUMBER', I7/)
      C     **************UPDATE CX(I,J) FOR BRO***************************
```

```
FORTRAN IV G LEVEL  21                  MAIN               DATE = 80315          11/07/24

C                          BRO IS THE RIGHT BRANCHING VARIABLE
  225 LX=BRO
      IX=LX/100
      JX=LX-IX*100
      CX(IX,JX)=2
      KT2(JX)=KT2(JX)+1
      FIX(JX)=0
      LQ1=LQ1-1
      IF (KT2(JX).LT.(M-1)) GO TO 270
      DO 255 I=1,M
         IF (CX(I,JX).EQ.2) GO TO 255
         CX(I,JX)=1
         NSX=NSX+1
         STX(NSX)=   (I*100+JX)+1000000
         FIX(JX)=1
         LQ1=LQ1+1
         FIXI(JX)=I
         GO TO 270
  255 CONTINUE
  270 LQ2=0
      LR2=0
      GO TO 283
  272 IF (ISTEP.EQ.0) GO TO 276
      PRINT 220,NOD
C     **************UPDATE CX(I,J) FOR BR1*********************************
C                          BR1 IS THE LEFT BRANCHING VARIABLE
  276 LQ2=0
      LR2=0
      LX=BR1
      IX=LX/100
      JX=LX-IX*100
      CX(IX,JX)=1
      FIX(JX)=1
      LQ1=LQ1+1
      DO 279 I=1,M
         IF (IX.EQ.I) GO TO 281
  279 CONTINUE
  281 FIXI(JX)=IX
  283 IF (ISTEP.NE.2) GO TO 303
  285 DO 295 I=1,M
         PRINT 290, I,(CX(I,J),J=1,N)
  290    FORMAT (/5X,'CX(I,J)',4X,'I=',I3,2X, 20I4/23X, 20I4)
  295 CONTINUE
      PRINT 297,(FIX(J),J=1,N)
  297 FORMAT (/5X,'FIX(J)',12X, 20I4/23X, 20I4)
C     **************APPLY CAPACITY RULE*******************************
C                          AND UPDATE CX(I,J) AND STX(INS).
  303 DO 307 K=1,P
         FLB(K)=0
  307 CONTINUE
  310 DO 2000 K=1,P
C                          FIND THE SUM OF MINIMUM D(I,J,K) OVER EACH J FOR A
C                          GIVEN K, I.E., MINSD= SUM OF MIND(J)
      MINSD=0
      DO 900 J=1,N
         IF(FIX(J).EQ.0) GO TO 350
C                          IF  FIX(J)=1, SET MIND(J)=D(I,J,K) FOR CX(I,J)=1
C                          AND MOVE TO NEXT COLUMN J
```

```
FORTRAN IV G LEVEL  21                  MAIN                DATE = 80315        11/07/24

            INDI=FIXI(J)
            MIND(J)=D(INDI,J,K)
            GO TO 800
      350      LK=0
               I=1
               MIND(J)=D(I,J,K)
C                                 SKIP D(I,J,K) WHEN CX(I,J)=2 & MOVE TO NEXT ROW I
      400      IF(CX(I,J).EQ.2) GO TO 600
      500      IF(D(I,J,K).LT.MIND(J)) MIND(J)=D(I,J,X)
               GO TO 700
      600      LK=LK+1
               IF(I.GT.LK) GO TO 700
               I=I+1
               MIND(J)=D(I,J,K)
               GO TO 750
      700      I=I+1
      750      IF(I.LE.M) GO TO 400
      800      MINSD=MINSD+MIND(J)
      900 CONTINUE
      910 IF (ISTEP.NE.2) GO TO 960
               PRINT 950, K, MINSD,(MIND(J),J=1,N)
      950      FORMAT ('0','K,MINSD,(MIND(J),J=1,N)',10I10,4(/,44X,8I10))
      960      IF (MINSD.EQ.0) GO TO 975
      965      IF( FLB(K).EQ.1) GO TO 975
      970      FLB(K)=1
      975      IF (IUNCAP.EQ.1) GO TO 2000
      978      IF (ICAPR.EQ.0) GO TO 2000
C                                 FIND BALANCE AVAILABLE CAPACITY IBALD FOR A GIVEN K
C                                 IF IBALD IS NEGATIVE, THEN BACKTRACK.
      980      IBALD=S(K)-MINSD
     1000 IF (IBALD.LT.0) GO TO 6200
            DO 1500 J=1,N
C                                 SKIP COLUMN J IF FIX(J)=1
               IF (FIX(J).EQ.1) GO TO 1500
            DO 1300 I=1,M
C                                 SKIP ROW I IF CX(I,J)=2
     1100      IF(CX(I,J).EQ.2) GO TO 1300
C                                 COMPUTE DIFFERENCE BETWEEN D(I,J,K) AND MIND(J).
C                                 IF IT IS MORE THAN AVAILABE BALANCE, SET CX(I,J)=2
     1200      IDIFD=D(I,J,K)-MIND(J)
               IF ((IDIFD-IBALD).LE.0)  GO TO 1300
               CX(I,J)=2
               NSX=NSX+1
               STX(NSX)=-(I*100+J)-1000000
C                                 LQ2 COUNTS THE NUMBER OF CX(I,J) VALUES SET EQUAL
C                                 TO 2 IN A CYCLE
               LQ2=LQ2+1
C                                 KT2(J) KEEPS AN ACCOUNT OF CX(I,J) VALUES SET EQUAL
C                                 TO 2 FOR COLUMN J
            KT2(J)=KT2(J)+1
C                                 FOR COLUMN J, IF ALL BUT ONE CX(I,J) VALUES ARE
C                                 EQUAL TO 2, SET THAT CX(I,J)=1 &  SET FIX(J)=1
               IF(KT2(J).LT.(M-1)) GO TO 1300
            DO 1250 LR=1,M
               IF(CX(LR,J).EQ.2) GO TO 1250
               CX(LR,J)=1
               NSX=NSX+1
               STX(NSX)=   (LR*100+J)+1000000
```

```
                FIX(J)=1
C                              LQ1 KEEPS AN ACCOUNT OF COLUMNS FOR WHICH FIX(J)=1
                LQ1=LQ1+1
C                              FIXI(J) SPECIFIES INDEX I FOR WHICH FIX(J)=1
                FIXI(J)=LR
                GO TO 1500
      1250 CONTINUE
      1300 CONTINUE
      1500 CONTINUE
      1800 IF (ISTEP.NE.2) GO TO 2000
           PRINT 1900, K, LQ2, LQ1
      1900 FORMAT ('0','K=',I3,'  LQ2=',I3, '  LQ1=', I3)
           DO 1930 I=1,M
                PRINT 290, I,(CX(I,J),J=1,N)
      1930 CONTINUE
           PRINT 297,  (FIX(J),J=1,N)
      2000 CONTINUE
C                              A CYCLE EXAMINES ALL THE FACILITIES.
C                              IF IN A CYCLE, THE CAPACITY RULE RESULTS IN SETTING
C                              ADDITIONAL CX(I,J) VALUES EQUAL TO 2, THEN REPEAT
C                              THE CYCLE. BUT IF FIX(J)=1 FOR ALL J, THEN DO NOT
C                              REPEAT THE CYCLE.
           IF (LQ1.EQ.N) GO TO 2400
           IF (LQ2.EQ.LR2) GO TO 2400
      2200 LR2=LQ2
           GO TO 310
C          **************SOLVE (LAGRANGIAN) RELAXED PROBLEM**************
C                            UPDATE VECTOR OF FACILITIES FLB(K) FOR COMPUTING
C                              C(I,J) MATRIX & LOWER BOUND. IT HAS VALUE 1 IF A
C                              FACILITY IS USED, OTHERWISE IT HAS 0 VALUE.
      2400 DO 3000 J=1,N
                IF (FIX(J).EQ.0) GO TO 3000
                INDI=FIXI(J)
           DO 2550 K=1,P
                IF (E(INDI,K).EQ.0) GO TO 2550
                IF (FLB(K).EQ.1) GO TO 2550
                FLB(K)=1
      2550 CONTINUE
      3000 CONTINUE
           IF (ISTEP.NE.2) GO TO 3150
           PRINT 3100, (FLB(K),K=1,P)
      3100 FORMAT('0','(FLB(K),K=1,P)  ',  20I4/16X,20I4)
C                              COMPUTE COST MATRIX C(I,J) FOR THE RELAXED PROBLEM
      3150 DO 3400 J=1,N
           DO 3300 I=1,M
                BSUM=0.0
           DO 3200 K=1,P
                IF (FLB(K).EQ.1) GO TO 3200
                IF (E(I,K).EQ.0) GO TO 3200
                BSUM=BSUM+(B(K) * (FLOAT(D(I,J,K))/ FLOAT(S(K))))
      3200 CONTINUE
      3250      C(I,J)=A(I,J)+BSUM
      3300 CONTINUE
      3400 CONTINUE
           IF (ISTEP.NE.2) GO TO 3445
           DO 3430 I=1,M
                PRINT 3420, I, (C(I,J),J=1,N)
      3420      FORMAT (/5X, 'C(I,J)',5X, 'I=', I3,2X, 5F15.4,
               1           6(/23X, 5F15.4))
            3430 CONTINUE
C                      FIND SUM OF MINIMUM C(I,J) VALUES OVER EACH J,
C                      I.E., MINSC=SUM OF MINC(J).
C                      IF FIX(J)=1, THEN MINC(J)=C(I,J) WHERE CX(I,J)=1
            3445 MINSC=0.0
                 DO 3900 J=1,N
                    IF (FIX(J).EQ.0) GO TO 3500
                    INDI=FIXI(J)
                    MINC(J)=C(INDI,J)
                    SOLX(J)=INDI
                    GO TO 3850
            3500    LK=0
                    I=1
C                      SKIP C(I,J) ELEMENT IF CX(I,J)=2 & MOVE TO NEXT I
            3550    IF (CX(I,J).EQ.2) GO TO 3700
                    IF ((I-LK).EQ.1) GO TO 3600
                    IF (C(I,J).GE.MINC(J)) GO TO 3750
            3600    MINC(J)=C(I,J)
                    IMIN=I
                    GO TO 3750
            3700    LK=LK+1
            3750    I=I+1
            3800    IF (I.LE.M) GO TO 3550
                    SOLX(J)=IMIN
            3850    MINSC=MINSC+MINC(J)
            3900 CONTINUE
                 IF (ISTEP.NE.2) GO TO 3940
                 DO 3920 J=1,N
                    PRINT 3910, J,MINC(J),SOLX(J)
            3910 FORMAT ('0','J,MINC(J),SOLX(J)', I5,F15.4,I6)
            3920 CONTINUE
C                      COMPUTE FIXED COST FC FOR LOWER BOUND
            3940 FC=0
                 DO 4000 K=1,P
                    IF (FLB(K).EQ.0) GO TO 4000
            3950    FC=FC+B(K)
            4000 CONTINUE
C        **************COMPUTE LOWER BOUND LOWB************************
            4050 LOWB=MINSC+FC
                 IF (ISTEP.EQ.0) GO TO 4150
                 PRINT 4120, MINSC, FC, LOWB
            4120 FORMAT ('0',' MINSC, FC, LOWB ', F15.4, I15, F15.4)
C                      COMPARE LOWER BOUND WITH BEST UPPER BOUND STAR
C                      BUBS WHICH EQUALS BUB/(1+EPS). IF LOWB IS
C                      GREATER THAN OR EQUAL TO BUBS, THEN BACKTRACK
            4150 IF (LOWB.GE.BUBS)GO TO 6200
C                      CHECK IF CURRENT SOLUTION SATISFIES CAPACITY
C                      CONSTRAINTS
            4200 IF (IUNCAP.EQ.1) GO TO 4420
            4210 DO 4400 K=1,P
                    NSUMD=0
                 DO 4300 J=1,N
                    IX=SOLX(J)
                    NSUMD=NSUMD+D(IX,J, K)
            4300 CONTINUE
                 IF (ISTEP.NE.2) GO TO 4320
                    PRINT 4310, K,NSUMD
```

```
FORTRAN IV G LEVEL  21                 MAIN                DATE = 80315          11/07/24
      4310    FORMAT ('0', 'K,NSUMD',2I10)
      4320    IF(NSUMD.LE.S(K)) GO TO 4400
              GO TO 5100
      4400 CONTINUE
    C      **************COMPUTE UPPER BOUND UPB IF CAPACITY CONSTRAINTS
    C                    ARE SATISFIED.
    C                    UPB=SUM OF A(I,J)+FIXED COST FCUB BASED ON
    C                    SOLUTION VECTOR SOLX(J)
    C                    VECTOR OF FACILITIES FOR UPPER BOUND FUB(K) HAS
    C                    VALUES 1 OR 0 BASED ON FACILILY USED OR OTHERWISE
      4420 DO 4450 K=1,P
              FUB(K)=0
      4450 CONTINUE
           NSUMA=0
           FCUB=0
      4500 DO 4650 J=1,N
              IX=SOLX(J)
              NSUMA=NSUMA+A(IX,J)
      4550 DO 4600 K=1,P
              IF(E(IX,K).EQ.0) GO TO 4600
              IF(FUB(K).EQ.1) GO TO 4600
              FUB(K)=1
              FCUB=FCUB+B(K)
      4600 CONTINUE
      4650 CONTINUE
           IF (ISTEP.NE.2) GO TO 4700
           PRINT 4660, (FUB(K),K=1,P)
      4660 FORMAT('0','(FUB(K),K=1,P)   ',  20I4/16X,20I4)
      4700 UPB=NSUMA+FCUB
      4708 IF (ISTEP.EQ.0) GO TO 4750
           PRINT 4710, NSUMA, FCUB, UPB, BUB, BUBS
      4710 FORMAT('0','NSUMA, FCUB, UPB, BUB, BUBS ',2I10, 3F15.4)
    C                    COMPARE UPPER BOUND WITH BEST UPPER BOUND
    C                    IF UPB IS LESS THAN BUB, SET IT AS BUB AND
    C                    NOTE THE SOLUTION
      4750 IF (UPB.GE.BUB) GO TO 5100
      4770 BUB=UPB
           BUBS= BUB/ (1.0+EPS)
           IBNOD=NOD
           PRINT 4780, IBNOD, BUB, BUBS
      4780 FORMAT ('0', 'IBNOD, BUB, BUBS', I10, 2F15.4)
           DO 4800 J=1,N
      4800 BSOLX(J)=SOLX(J)
           DO 4850 K=1,P
      4850 BSOLY(K)=FUB(K)
    C      **************COMPARE LOWB WITH BUBS. IF LOWB IS GREATER
    C                    THAN OR EQUAL TO BUBS, THEN BACKTRACK
      4900 IF (LOWB.GE.BUBS)GO TO 6200
    C      **************IF FIX(J) VALUES ARE 1 FOR EACH J, THEN BACKTRACK
      5100 IF (LQ1.EQ.N) GO TO 6200
    C      **************APPLY THE BOUNDING RULE*************************
    C                    IF THE DIFFERENCE BETWEEN C(I,J) AND MINC(J) IS
    C                    GREATER THAN THE DIFFERENCE BETWEEN BUBS AND
    C                    LOWB, THEN CX(I,J)=2
    C      **************APPLY BRANCHING RULE AND FIND BR1, THE NEXT
    C                    VARIABLE FOR LEFT BRANCHING.
    C                    FIND NMINC(J), THE NEXT HIGHER VALUE THAN MINC(J)
    C                    AND DIFBR(J), THE DIFFERENCE BETWEEN THEM.
```

```
FORTRAN IV G LEVEL  21                 MAIN               DATE = 80315         11/07/24
             DBOUND=BUBS-LOWB
       5200 DO 5250 J=1,N
               NMINC(J)=0.0
               DIFBR(J)=0.0
       5250 CONTINUE
             DO 5600 J=1,N
      C                          SKIP TO NEXT J IF FIX(J)=1
               IF (FIX(J).EQ.1) GO TO 5600
               LK=0
               I=1
      C                          SKIP C(I,J) IF CX(I,J)=2 & MOVE TO NEXT I
       5300    IF (CX(I,J).EQ.2) GO TO 5350
               IF (I.EQ.SOLX(J)) GO TO 5350
               IF ((C(I,J)-MINC(J)).GT.DBOUND) GO TO 5330
               IF ((I-LK).EQ.1) GO TO 5320
               IF (C(I,J).GE.NMINC(J)) GO TO 5400
       5320    NMINC(J)=C(I,J)
               GO TO 5400
       5330    CX(I,J)=2
               NSX=NSX+1
               STX(NSX)=-(I*100+J)-1000000
               KT2(J)=KT2(J)+1
               IF(KT2(J).LT.(M-1)) GO TO 5350
               INDI=SOLX(J)
               CX(INDI,J)=1
               NSX=NSX+1
               STX(NSX)= (INDI*100+J)+1000000
               FIX(J)=1
               LQ1=LQ1+1
               FIXI(J)=INDI
               GO TO 5600
       5350    LK=LK+1
       5400    I=I+1
               IF(I.LE.M) GO TO 5300
       5500    DIFBR(J)=NMINC(J)-MINC(J)
       5600 CONTINUE
             IF (ISTEP.NE.2) GO TO 5650
             DO 5620 I=1,M
                PRINT 290, I,(CX(I,J),J=1,N)
       5620 CONTINUE
             PRINT 297,   (FIX(J),J=1,N)
      C                          IF FIX(J)=1 FOR ALL J, THEN BACKTRACK.
       5650 IF (LQ1.EQ.N) GO TO 6200
      C                          FIND MAXDIF, THE MAXIMUM DIFFERENCE DIFBR(J)
             LF=0
             DO 5800 J=1,N
                IF (FIX(J).EQ.1) GO TO 5690
                IF ((J-LF).EQ.1) GO TO 5660
                IF (DIFBR(J).LT.MAXDIF) GO TO 5800
       5660     MAXDIF=DIFBR(J)
                LJ=J
                GO TO 5800
       5690 LF=LF+1
       5800 CONTINUE
             IF (ISTEP.NE.2) GO TO 5840
             DO 5820 J=1,N
                IF (FIX(J).EQ.1) GO TO 5820
                PRINT 5810, J, NMINC(J), MINC(J), DIFBR(J)
```

```
 5810 FORMAT ('0','J,NMINC(J),MINC(J),DIFBR(J)', I5,3F15.4)             00004963
 5820 CONTINUE                                                          00004966
C     **************BRANCHING VARIABLE BR1 CORRESPONDS TO MAXDIF*****   00004970
 5840 DO 5900 J=1,N                                                     00004980
         IF (J.NE.LJ) GO TO 5900                                        00004990
 5850    BR1=SOLX(J)*100+J                                              00005000
      IF (ISTEP.EQ.0) GO TO 6020                                        00005010
         PRINT 5880,  BR1                                               00005020
 5880    FORMAT('0',' BR1',I10)                                         00005030
         GO TO 6020                                                     00005040
 5900 CONTINUE                                                          00005050
C     **************UPDATE STX(INS) AND NSX************************     00005060
C                   NSX REPRESENTS THE NUMBER OF VARIABLES IN STX(INS)  00005070
 6020 NSX=NSX+1                                                         00005090
 6040 STX(NSX)=BR1                                                      00005100
      IF (ISTEP.NE.2) GO TO 6100                                        00005150
      PRINT 6088, (STX(INS), INS=1,NSX)                                 00005160
 6088 FORMAT('0',' STX(INS)', 10I10, 122(/, 10X,10I10))                 00005170
C     **************MOVE TO THE NEXT (LEFT BRANCH) NODE AND APPLY       00005220
C                   CAPACITY RULE                                       00005230
 6100 NOD=NOD+1                                                         00005240
 6110 IF (ET.EQ.0.0) GO TO 6150                                         00005242
      IF(INSET.EQ.1) GO TO 6147                                         00005244
      IF (INET.EQ.1) GO TO 6150                                         00005246
      CALL TIMET(INT)                                                   00005248
      ELTN=(INT-IT0)*26.04E-6                                           00005250
      IF (ELTN.LT.ET) GO TO 6150                                        00005253
 6120 PRINT 6125, NOD, ELTN, BUB, BUBS, IBNOD                           00005256
 6125 FORMAT ('0', 'WAS AT NODE',I6, ' AT ELAPSED TIME =', F10.4,       00005260
     1         ' SECONDS.',/1X, '  BUB=',F15.4, '  BUBS=',F15.4,        00005263
     2         '  AT NODE=',I7)                                         00005266
      IBUB=BUB                                                          00005267
      IF (IBUB.EQ.9999999) GO TO 6146                                   00005268
 6130 PRINT 6135, (BSOLX(J),J=1,N)                                      00005270
 6135 FORMAT('0', 'SOLUTION CORRESPONDING TO BUB IS',  //1X,            00005273
     1        '(BSOLX(J), J=1,N)',10I8,3(/18X,10I8))                    00005276
 6140 PRINT 6145, (BSOLY(K), K=1,P)                                     00005280
 6145 FORMAT(/1X,'(BSOLY(K), K=1,P)',10I8,2(/18X, 10I8))                00005290
 6146 INET=1                                                            00005292
      INIS=ISTEP                                                        00005294
      INSET=1                                                           00005296
      ISTEP=2                                                           00005298
      GO TO 6150                                                        00005300
 6147 ISTEP=INIS                                                        00005302
      INSET=0                                                           00005304
 6150 GO TO 272                                                         00005306
C     **************END IF AT THE ROOT NODE*************************    00005308
 6200 IF (NSX.EQ.0) GO TO 8100                                          00005310
 6220 IF ( IABS(STX(NSX)).GT.1000000) GO TO 6500                        00005320
 6250 BRO=STX(NSX)                                                      00005330
 6270 STX(NSX)=-BRO-1000000                                             00005340
      IF (ISTEP.EQ.0) GO TO 6308                                        00005390
      PRINT 6305,  BRO                                                  00005400
 6305 FORMAT('0', 'BRO  ',I10)                                          00005410
 6308 IF (ISTEP.NE.2) GO TO 6330                                        00005420
      PRINT 6088, (STX(INS), INS=1,NSX)                                 00005430
C     **************MOVE TO THE NEXT (RIGHT BRANCH) NODE AND APPLY      00005490
C                   CAPACITY RULE                                       00005500
```

```
      6330 NOD=NOD+1
      6410 IF (ET.EQ.0.0) GO TO 6450
           IF (INSET.EQ.1) GO TO 6445
           IF (INET.EQ.1) GO TO 6450
           CALL TIMET(INT)
           ELTN=(INT-ITO)*26.04E-6
           IF (ELTN.LT.ET) GO TO 6450
      6420 PRINT 6125, NOD, ELTN, BUB, BUBS, IBNOD
           IBUB=BUB
           IF (IBUB.EQ.9999999) GO TO 6442
      6430 PRINT 6135, (BSOLX(J),J=1,N)
      6440 PRINT 6145, (BSOLY(K), K=1,P)
      6442 INET=1
           INIS=ISTEP
           INSET=1
           ISTEP=2
           GO TO 6450
      6445 ISTEP=INIS
           INSET=0
      6450 GO TO 210
      6500 IF ( STX(NSX).GT.1000000) GO TO 6520
           LX=-STX(NSX)-1000000
           IX=LX/100
           JX=LX-IX*100
           CX(IX,JX)=0
           KT2(JX)=KT2(JX)-1
           GO TO 6550
      6520 LX= STX(NSX)-1000000
           IX=LX/100
           JX=LX-IX*100
           CX(IX,JX)=0
           FIX(JX)=0
           LQ1=LQ1-1
      6550 NSX=NSX-1
           GO TO 6200
     C     **************PRINT THE OUTPUT**********************************
      8100 IBUB=BUB
           CALL TIMET(IT1)
           ELT1=(IT1-ITO)*26.04E-6
           PRINT 8105, ELT1
      8105 FORMAT ('0',///1X, 'ELAPSED TIME IN SECONDS=', F15.8)
           PRINT 8120, NOD
      8120 FORMAT ( '0','TOTAL NUMBER OF NODES EXPLORED =',I3)
           IF (IBUB.EQ.9999999) GO TO 8350
      8130 PRINT 8150
      8150 FORMAT ('0', 'NOTE: 1.  FOLLOWING X(I,J) VARIABLES SHOW DESIGN',
          1          ' I TO WHICH ACTIVITY J IS ASSIGNED FOR J=1 TO N.',
          2          /7X, '2.  IF EPSILON EPS WAS ASSIGNED A POSITIVE',
          3          ' (NON-ZERO) VALUE, THE SOLUTION MAY BE SUBOPTIMAL.',/)
      8180 PRINT 8200, (BSOLX(J),J=1,N)
      8200 FORMAT('0',T55,  'OPTIMAL SOLUTION',/1X,T55,
          1          '----------------',//1X, 'X(I,J) WITH VALUE 1:',10I8,
          2          3(/,21X,10I8))
      8220 PRINT 8250, (BSOLY(K), K=1,P)
      8250 FORMAT ('0',      'Y(K) VALUES:', 8X, 10I8, 2(/,21X,10I8))
      8280 PRINT 8300,IBUB
      8300 FORMAT ('0',  'OPTIMAL VALUE OF OBJECTIVE FUNCTION:', I15///)
           GO TO 8500
```

```
FORTRAN IV G LEVEL  21                 MAIN               DATE = 80315         11/07/24

 0482          8350 PRINT 8400                                                         00005960
 0483          8400 FORMAT ('0', ' PROBLEM DOES NOT HAVE A FEASIBLE SOLUTION',         00005970
                   1        /1X,    ' BECAUSE THE CAPACITY CONSTRAINTS CANNOT',        00005980
                   2        /1X, ' BE SATISFIED.',/)                                   00005990
 0484          8500 PRINT 8550                                                         00006000
 0485          8550 FORMAT ('0','*****NORMAL END OF JOB*****',/)                       00006010
 0486          8600 STOP                                                               00006020
 0487               END                                                                00006030
```

APPENDIX B

DETAILED PRINTOUT FOR A TEST PROBLEM
(TEST PROBLEM WITH m=5, n=4, AND p=8)

```
OPTIONS SELECTED :  IINPT=1  ICAPR=1  ISTEP=2  IUNCAP=0  EPS= 0.0      ET=     0.0

                                        INPUT DATA
                                        ----- ----

                                NUMBER OF DESIGNS (M)=        5
                                NUMBER OF ACTIVITIES (N)=     4
                                NUMBER OF FACILITIES (P)=     8

VARIABLE COST MATRIX  A(I,J)

 I=  1          194951      218871      155094      104056
 I=  2          235277      272087      143264      138641
 I=  3          196138      220718      160399      107481
 I=  4          198751      224042      167046      112445
 I=  5          190873      229169      128361      112498

FIXED COST VECTOR  B(K)

                14000       14000       14000       14000       14000       29000       19000       31000

CAPACITY LIMIT VECTOR  S(K)

                  350         350         350         200         200         700         500         800

CAPACITY USAGE MATRIX  D(I,J,K)

 K=  1
```

```
I=  1          80        90       100       180
I=  2           0         0         0         0
I=  3          80        90       100       180
I=  4          40        30        50       120
I=  5           0         0         0         0

K=  2

I=  1          80        90       100       180
I=  2           0         0         0         0
I=  3          80        90       100       180
I=  4          40        30        50       120
I=  5           0         0         0         0

K=  3

I=  1          80        90       100       180
I=  2           0         0         0         0
I=  3          80        90       100       180
I=  4          40        30        50       120
I=  5           0         0         0         0

K=  4

I=  1          80        90       100       180
I=  2           0         0         0         0
I=  3          80        90       100       180
I=  4          40        30        50       120
I=  5          80        90       100       180

K=  5

I=  1          80        90       100       180
I=  2           0         0         0         0
I=  3          80        90       100       180
```

```
I=  4         40        30        50       120
I=  5         80        90       100       180

K=  6

I=  1        240       180       300       450
I=  2          0         0         0         0
I=  3        120        90       150       360
I=  4          0         0         0         0
I=  5         40        30        50       120

K=  7

I=  1        160       180       200       360
I=  2          0         0         0         0
I=  3         80        60       100       240
I=  4          0         0         0         0
I=  5        160       180       200       360

K=  8

I=  1        200       150       250       600
I=  2         40        30        50       180
I=  3          0         0         0         0
I=  4          0         0         0         0
I=  5        120        90       150       300
```

DESIGN-FACILITY MATRIX E(I,K)

```
I=  1          1         1         1         1         1         1         1         1

I=  2          0         0         0         0         0         0         0         1

I=  3          1         1         1         1         1         1         1         0

I=  4          1         1         1         1         1         0         0         0
```

```
I=  5           0         0         0         1         1         1         1         1


                            DETAILED LISTING OF STEPS


    NODE NUMBER       1
    ===================

    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0                                        Step 1
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         1       0       0       0       0       0      Step 2 - - k=1
K=  1  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         2       0       0       0       0       0      Step 2 - - k=2
K=  2  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         3       0       0       0       0       0      Step 2 - - k=3
K=  3  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         4         0         0         0         0         0            Step 2 - - k=4
K=  4  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         5         0         0         0         0         0            Step 2 - - k=5
K=  5  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         6         0         0         0         0         0            Step 2 - - k=6
K=  6  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)         7         0         0         0         0         0            Step 2 - - k=7
K=  7  LQ2=  0  LQ1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
K,MINSD,(MIND(J),J=1,N)          8         0         0         0         0         0        Step 2 - - k=8
K=  8  LO2=  0  LO1=  0
    CX(I,J)     I=  1     0   0   0   0
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   0   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
(FLB(K),K=1,P)     0   0   0   0   0   0   0   0                                             Step 3
    C(I,J)      I=  1      239180.9375     262123.4375     210381.4375     205785.9375
    C(I,J)      I=  2      236826.9375     273249.4375     145201.5000     145615.9375
    C(I,J)      I=  3      224777.9375     249997.9375     196198.9375     177800.9375
    C(I,J)      I=  4      209150.9375     231841.9375     180045.9375     143644.9375
    C(I,J)      I=  5      214402.9375     253296.4375     157773.4375     167802.9375
J,MINC(J),SOLX(J)    1     209150.9375      4
J,MINC(J),SOLX(J)    2     231841.9375      4
J,MINC(J),SOLX(J)    3     145201.5000      2
J,MINC(J),SOLX(J)    4     143644.9375      4
 P'NSC, FC, LOWB     72°839.3125               0     72983°.3125                            Step 4
K,NSUMD          1       190                                                                Step 6
K,NSUMD          2       190
K,NSUMD          3       190
K,NSUMD          4       190
K,NSUMD          5       190
K,NSUMD          6         0
K,NSUMD          7         0
K,NSUMD          8        50
(FUB(K),K=1,P)     1   1   1   1   1   0   0   1                                             Step 7
NSUMA, FCUB, UPB, BUB, RUBS     678502     101000     779502.0000     9999999.0000     9999999.0000
IBNOD, RUB, BUBS          1     779502.0000     779502.0000
```

```
    CX(I,J)     I=  1     0   0   2   2                                                      Step 11
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   2   0
    CX(I,J)     I=  4     0   0   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   0   0   0
J,NMINC(J),MINC(J),DIFBR(J)     1    214402.9375    209150.9375      5252.0000
J,NMINC(J),MINC(J),DIFBR(J)     2    249997.9375    231841.9375     18156.0000
J,NMINC(J),MINC(J),DIFBR(J)     3    157773.4375    145201.5000     12571.9375
J,NMINC(J),MINC(J),DIFBR(J)     4    145615.9375    143644.9375      1971.0000
 BR1       402
 STX(INS)  -1000103  -1000303  -1000104       402

    NODE NUMBER        2
    ===================

    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   2   0
    CX(I,J)     I=  4     0   1   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   1   0   0
K,MINSD,(MIND(J),J=1,N)          1       30         0        30         0         0          Step 2 - - k=1
K=  1  LQ2=  0  LQ1=  1
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   2   0
    CX(I,J)     I=  4     0   1   0   0
    CX(I,J)     I=  5     0   0   0   0
    FIX(J)                0   1   0   0
K,MINSD,(MIND(J),J=1,N)          2       30         0        30         0         0          Step 2 - - k=2
K=  2  LQ2=  0  LQ1=  1
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)    I=  3     0   0   2   0
    CX(I,J)    I=  4     0   1   0   0
    CX(I,J)    I=  5     0   0   0   0
    FIX(J)               0   1   0   0
K,MINSD,(MIND(J),J=1,N)        3        30        0        30        0        0          Step 2 - - k=3
K=  3  LQ2=  0  LQ1=  1
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     0   0   0   0
    CX(I,J)    I=  3     0   0   2   0
    CX(I,J)    I=  4     0   1   0   0
    CX(I,J)    I=  5     0   0   0   0
    FIX(J)               0   1   0   0
K,MINSD,(MIND(J),J=1,N)        4        30        0        30        0        0          Step 2 - - k=4
K=  4  LQ2=  2  LQ1=  1
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     0   0   0   0
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4     0   1   0   0
    CX(I,J)    I=  5     0   0   0   2
    FIX(J)               0   1   0   0
K,MINSD,(MIND(J),J=1,N)        5        30        0        30        0        0          Step 2 - - k=5
K=  5  LQ2=  2  LQ1=  1
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     0   0   0   0
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4     0   1   0   0
    CX(I,J)    I=  5     0   0   0   2
    FIX(J)               0   1   0   0
K,MINSD,(MIND(J),J=1,N)        6        0         0        0         0        0          Step 2 - - k=6
K=  6  LQ2=  2  LQ1=  1
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     0   0   0   0
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)       7       0       0       0       0       0      Step 2 - - k=7
K=  7  LQ2=  2  LQ1=  1
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)       8       0       0       0       0       0      Step 2 - - k=8
K=  8  LQ2=  2  LQ1=  1
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)       1      30       0      30       0       0      Step 2 - - k=1
K=  1  LQ2=  2  LQ1=  1                                                       (Second Cycle)
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)       2      30       0      30       0       0      Step 2 - - k=2
K=  2  LQ2=  2  LQ1=  1                                                       (Second Cycle)
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)         3        30         0        30         0         0          Step 2 - - k=3
K=  3  LQ2=  2  LQ1=  1                                                                      (Second Cycle)
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)         4        30         0        30         0         0          Step 2 - - k=4
K=  4  LQ2=  2  LQ1=  1                                                                      (Second Cycle)
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)         5        30         0        30         0         0          Step 2 - - k=5
K=  5  LQ2=  2  LQ1=  1                                                                      (Second Cycle)
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)              0   1   0   0
K,MINSD,(MIND(J),J=1,N)         6         0         0         0         0         0          Step 2 - - k=6
K=  6  LQ2=  2  LQ1=  1                                                                      (Second Cycle)
    CX(I,J)    I=  1    0   0   2   2
    CX(I,J)    I=  2    0   0   0   0
    CX(I,J)    I=  3    0   0   2   2
    CX(I,J)    I=  4    0   1   0   0
    CX(I,J)    I=  5    0   0   0   2
    FIX(J)                0   1   0   0
K,MINSC,(MIND(J),J=1,N)         7          0          0          0          0          0     Step 2 - - k=7
K=  7  LQ2=  2  LQ1=  1                                                                    (Second Cycle)
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   2   2
    CX(I,J)     I=  4     0   1   0   0
    CX(I,J)     I=  5     0   0   0   2
    FIX(J)                0   1   0   0
K,MINSD,(MIND(J),J=1,N)         8          0          0          0          0          0     Step 2 - - k=8
K=  8  LQ2=  2  LQ1=  1                                                                    (Second Cycle)
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     0   0   0   0
    CX(I,J)     I=  3     0   0   2   2
    CX(I,J)     I=  4     0   1   0   0
    CX(I,J)     I=  5     0   0   0   2
    FIX(J)                0   1   0   0
(FLB(K),K=1,P)       1    1    1    1    1    0    0    0                                     Step 3
    C(I,J)      I=  1      218380.9375    238723.4375    184381.4375    158985.9375
    C(I,J)      I=  2      236826.9375    273249.4375    145201.5000    145615.9375
    C(I,J)      I=  3      203977.9375    226597.9375    170198.9375    131000.9375
    C(I,J)      I=  4      198751.0000    224042.0000    1670[illegible]6.0000    112445.0000
    C(I,J)      I=  5      203202.9375    240696.4375    143773.4375    142602.9375
J,MINC(J),SOLX(J)     1     198751.0000      4
J,MINC(J),SOLX(J)     2     224042.0000      4
J,MINC(J),SOLX(J)     3     143773.4375      5
J,MINC(J),SOLX(J)     4     112445.0000      4
 MINSC, FC, LOWB      679011.4375          70000     749011.4375                          Step 4
K,NSUMD           1         190                                                           Step 6
K,NSUMD           2         190
K,NSUMD           3         190
K,NSUMD           4         290
    CX(I,J)     I=  1     0   0   2   2                                                   Step 11
    CX(I,J)     I=  2     2   0   0   2                                                           Step 11 (continued)
    CX(I,J)     I=  3     0   0   2   2
    CX(I,J)     I=  4     0   1   0   1
    CX(I,J)     I=  5     0   0   0   2
    FIX(J)                0   1   0   1
J,NMINC(J),MINC(J),DIFBR(J)     1     203202.9375     198751.0000      4451.9375
J,NMINC(J),MINC(J),DIFBR(J)     3     145201.5000     143773.4375      1428.0625
BR1        401
STX(INS)  -1000103  -1000303  -1000104       402  -1000304  -1000504  -1000201  -1000204   1000404       401


    NODE NUMBER       3
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     2   0   0   2
    CX(I,J)     I=  3     0   0   2   2
    CX(I,J)     I=  4     1   1   0   1
    CX(I,J)     I=  5     0   0   0   2
    FIX(J)                1   1   0   1
K,MINSD,(MIND(J),J=1,N)         1       190        40        30         0       120            Step 2 - - k=1
K=  1  LQ2=  0  LQ1=  3
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     2   0   0   2
    CX(I,J)     I=  3     0   0   2   2
    CX(I,J)     I=  4     1   1   0   1
    CX(I,J)     I=  5     0   0   0   2
    FIX(J)                1   1   0   1
K,MINSD,(MIND(J),J=1,N)         2       190        40        30         0       120            Step 2 - - k=2
K=  2  LQ2=  0  LQ1=  3
    CX(I,J)     I=  1     0   0   2   2
    CX(I,J)     I=  2     2   0   0   2
    CX(I,J)     I=  3     0   0   2   2
    CX(I,J)     I=  4     1   1   0   1
    CX(I,J)     I=  5     0   0   0   2
    FIX(J)                 1    1    0    1
K,MINSD,(MIND(J),J=1,N)         3        190        40        30        0        120        Step 2 - - k=3
K=  3  LC2=  0  LQ1=  3
    CX(I,J)     I=  1      0    0    2    2
    CX(I,J)     I=  2      2    0    0    2
    CX(I,J)     I=  3      0    0    2    2
    CX(I,J)     I=  4      1    1    0    1
    CX(I,J)     I=  5      0    0    0    2
    FIX(J)                 1    1    0    1
K,MINSD,(MIND(J),J=1,N)         4        190        40        30        0        120        Step 2 - - k=4
K=  4  LQ2=  2  LQ1=  4
    CX(I,J)     I=  1      0    0    2    2
    CX(I,J)     I=  2      2    0    1    2
    CX(I,J)     I=  3      0    0    2    2
    CX(I,J)     I=  4      1    1    2    1
    CX(I,J)     I=  5      0    0    2    2
    FIX(J)                 1    1    1    1
K,MINSD,(MIND(J),J=1,N)         5        190        40        30        0        120        Step 2 - - k=5
K=  5  LQ2=  2  LQ1=  4
    CX(I,J)     I=  1      0    0    2    2
    CX(I,J)     I=  2      2    0    1    2
    CX(I,J)     I=  3      0    0    2    2
    CX(I,J)     I=  4      1    1    2    1
    CX(I,J)     I=  5      0    0    2    2
    FIX(J)                 1    1    1    1
K,MINSD,(MIND(J),J=1,N)         6          0         0         0        0          0        Step 2 - - k=6
K=  6  LQ2=  2  LQ1=  4
    CX(I,J)     I=  1      0    0    2    2
    CX(I,J)     I=  2      2    0    1    2
    CX(I,J)     I=  3      0    0    2    2
    CX(I,J)     I=  4      1    1    2    1
    CX(I,J)     I=  5      0    0    2    2
    FIX(J)                 1    1    1    1
```

```
K,MINSD,(MIND(J),J=1,N)        7         0         0         0         0         0          Step 2 - - k=7
K=  7  LQ2=  2  LQ1=  4
   CX(I,J)    I=  1      0   0   2   2
   CX(I,J)    I=  2      2   0   1   2
   CX(I,J)    I=  3      0   0   2   2
   CX(I,J)    I=  4      1   1   2   1
   CX(I,J)    I=  5      0   0   2   2
   FIX(J)                1   1   1   1
K,MINSD,(MIND(J),J=1,N)        8        50         0         0        50         0          Step 2 - - k=8
K=  8  LQ2=  2  LQ1=  4
   CX(I,J)    I=  1      0   0   2   2
   CX(I,J)    I=  2      2   0   1   2
   CX(I,J)    I=  3      0   0   2   2
   CX(I,J)    I=  4      1   1   2   1
   CX(I,J)    I=  5      0   0   2   2
   FIX(J)                1   1   1   1
(FLB(K),K=1,P)      1   1   1   1   1   0   0   1                                          Step 3
   C(I,J)     I=  1      210630.9375     232910.9375     174693.9375     135735.9375
   C(I,J)     I=  2      235277.0000     272087.0000     143264.0000     138641.0000
   C(I,J)     I=  3      203977.9375     226597.9375     170198.9375     131000.9375
   C(I,J)     I=  4      198751.0000     224042.0000     167046.0000     112445.0000
   C(I,J)     I=  5      198552.9375     237208.9375     137960.9375     130977.9375
J,MINC(J),SOLX(J)     1     198[illegible]      4
J,MINC(J),SOLX(J)     2     224042.0000      4
J,MINC(J),SOLX(J)     3     143264.0000      2
J,MINC(J),SOLX(J)     4     112445.0000      4
 MINSC, FC, LOWB     678502.0000          101000     779502.0000                            Step 4
BRO          401                                                                            Step 12
 STX(INS)  -1000103  -1000303  -1000104       402  -1000304  -1000504  -1000201  -1000204   1000404  -1000401


     NODE NUMBER       4
     ====================

    CX(I,J)    I=  1      0   0   2   2
    CX(I,J)    I=  2     2   0   0   2
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4     2   1   0   1
    CX(I,J)    I=  5     0   0   0   2
    FIX(J)               0   1   0   1
K,MINSD,(MIND(J),J=1,N)        1      150        0        30        0       120                Step 2 - - k=1
K=  1  LQ2=  0  LQ1=  2
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     2   0   0   2
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4     2   1   0   1
    CX(I,J)    I=  5     0   0   0   2
    FIX(J)               0   1   0   1
K,MINSD,(MIND(J),J=1,N)        2      150        0        30        0       120                Step 2 - - k=2
K=  2  LQ2=  0  LQ1=  2
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     2   0   0   2
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4     2   1   0   1
    CX(I,J)    I=  5     0   0   0   2
    FIX(J)               0   1   0   1
K,MINSD,(MIND(J),J=1,N)        3      150        0        30        0       120                Step 2 - - k=3
K=  3  LQ2=  0  LQ1=  2
    CX(I,J)    I=  1     0   0   2   2
    CX(I,J)    I=  2     2   0   0   2
    CX(I,J)    I=  3     0   0   2   2
    CX(I,J)    I=  4     2   1   0   1
    CX(I,J)    I=  5     0   0   0   2
    FIX(J)               0   1   0   1
K,MINSD,(MIND(J),J=1,N)        4      230       80        30        0       120                Step 2 - - k=4
BRO        402                                                                                 Step 12
 STX(INS)  -1000103  -1000303  -1000104  -1000402
```

(Last Page of Printout)

```
    C(I,J)      I=  1     204550.9375    229670.9375    167093.9375    125655.9375
    C(I,J)      I=  2     235277.0000    272087.0000    143264.0000    138641.0000
    C(I,J)      I=  3     205737.9375    231517.9375    172398.9375    129080.9375
    C(I,J)      I=  4     203550.9375    227641.9375    173045.9375    126844.9375
    C(I,J)      I=  5     190873.0000    229169.0000    128361.0000    112498.0000
J,MINC(J),SOLX(J)    1    190873.0000     5
J,MIN[illegible](J),SOLX(J)    2    229169.0[illegible]00     5
J,MINC(J),SOLX(J)    3    128361.0000     5
J,MINC(J),SOLX(J)    4    138641.0000     2
 MINSC, FC, LOWB     687044.0000          106000     793044.0000


ELAPSED TIME IN SECONDS=     2.03278542
TOTAL NUMBER OF NODES EXPLORED =  9
NOTE: 1.  FOLLOWING X(I,J) VARIABLES SHOW DESIGN I TO WHICH ACTIVITY J IS ASSIGNED FOR J=1 TO N.
      2.  IF EPSILON EPS WAS ASSIGNED A POSITIVE (NON-ZERO) VALUE, THE SOLUTION MAY BE SUBOPTIMAL.

                                    OPTIMAL SOLUTION
                                    ----------------
X(I,J) WITH VALUE 1:       4      4      2      4
Y(K) VALUES:               1      1      1      1      1      0      0      1
OPTIMAL VALUE OF OBJECTIVE FUNCTION:         779502


******NORMAL [illegible] [illegible] JOB******
```

THE GEORGE WASHINGTON UNIVERSITY
Program in Logistics
Distribution List for Technical Papers

The George Washington University
  Office of Sponsored Research
  Gelman Library
  Vice President H. F. Bright
  Dean Harold Liebowitz
  Dean Henry Solomon

ONR
  Chief of Naval Research
   (Codes 200, 434)
  Resident Representative

OPNAV
  OP-40
  DCNO, Logistics
  Navy Dept Library
  NAVDATA Automation Cmd

Naval Aviation Integrated Log Support

NARDAC Tech Library

Naval Electronics Lab Library

Naval Facilities Eng Cmd Tech Library

Naval Ordnance Station
  Louisville, Ky.
  Indian Head, Md.

Naval Ordnance Sys Cmd Library

Naval Research Branch Office
  Boston
  Chicago
  New York
  Pasadena
  San Francisco

Naval Ship Eng Center
  Philadelphia, Pa.

Naval Ship Res & Dev Center

Naval Sea Systems Command
  PMS 30611
  Tech Library
  Code 073

Naval Supply Systems Command
  Library
  Operations and Inventory Analysis

Naval War College Library
  Newport

BUPERS Tech Library

FMSO

USN Ammo Depot Earle

USN Postgrad School Monterey
  Library
  Dr Jack R. Borsting
  Prof C. R. Jones

US Coast Guard Academy
  Capt Jimmie D. Woods

US Marine Corps
  Commandant
  Deputy Chief of Staff, R&D

Marine Corps School Quantico
  Landing Force Dev Ctr
  Logistics Officer

Armed Forces Industrial College

Armed Forces Staff College

Army War College Library
  Carlisle Barracks

Army Cmd & Gen Staff College

Army Logistics Mgt Center
  Fort Lee

Commanding Officer, USALDSRA
  New Cumberland Army Depot

Army Inventory Res Ofc
  Philadelphia

Army Trans Material Cmd
  TCMAC-ASDT

Air Force Headquarters
  AFADS-3
  LEXY
  SAF/ALG

Griffiss Air Force Base
  Reliability Analysis Center

Gunter Air Force Base
  AFLMC/XR

Maxwell Air Force Base Library

Wright-Patterson Air Force Base
  AFLC/OA
  Research Sch Log
  AFALD/XR

Defense Technical Info Center

National Academy of Sciences
  Maritime Transportation Res Bd Lib

National Bureau of Standards
  Dr B. H. Colvin
  Dr Joan Rosenblatt

National Science Foundation

National Security Agency

Weapons Systems Evaluation Group

British Navy Staff

National Defense Hdqtrs, Ottawa
  Logistics, OR Analysis Estab

American Power Jet Co
  George Chernowitz

General Dynamics, Pomona

General Research Corp
  Library

Logistics Management Institute
  Dr Murray A. Geisler

Rand Corporation
  Library
  Mr William P. Hutzler

Carnegie-Mellon University
  Dean H. A. Simon
  Prof G. Thompson

Case Western Reserve University
  Prof B. V. Dean
  Prof M. Mesarovic

Cornell University
  Prof R. E. Bechhofer
  Prof R. W. Conway
  Prof Andrew Schultz, Jr.

Cowles Foundation for Research
  in Economics
  Prof Martin Shubik

Florida State University
  Prof R. A. Bradley

Harvard University
  Prof W. G. Cochran
  Prof Arthur Schleifer, Jr.

Princeton University
  Prof A. W. Tucker
  Prof J. W. Tukey
  Prof Geoffrey S. Watson

Purdue University
  Prof S. S. Gupta
  Prof H. Rubin
  Prof Andrew Whinston

Stanford University
  Prof T. W. Anderson
  Prof Kenneth Arrow
  Prof G. B. Dantzig
  Prof F. S. Hillier
  Prof D. L. Iglehart
  Prof Samuel Karlin
  Prof G. J. Lieberman
  Prof Herbert Solomon
  Prof A. F. Veinott, Jr.

University of California, Berkeley
  Prof R. E. Barlow
  Prof D. Gale
  Prof Jack Kiefer

University of California, Los Angeles
  Prof R. R. O'Neill

University of North Carolina
  Prof W. L. Smith
  Prof M. R. Leadbetter

University of Pennsylvania
  Prof Russell Ackoff

University of Texas
  Institute for Computing Science
  and Computer Applications

Yale University
  Prof F. J. Anscombe
  Prof H. Scarf

Prof Z. W. Birnbaum
  University of Washington

Prof B. H. Bissinger
  The Pennsylvania State University

Prof Seth Bonder
  University of Michigan

Prof G. E. Box
  University of Wisconsin

Dr Jerome Bracken
  Institute for Defense Analyses

Continued

Prof A. Charnes
University of Texas

Prof H. Chernoff
Mass Institute of Technology

Prof Arthur Cohen
Rutgers - The State University

Mr Wallace M. Cohen
US General Accounting Office

Prof C. Derman
Columbia University

Prof Masao Fukushima
Kyoto University

Prof Saul I. Gass
University of Maryland

Dr Donald P. Gaver
Carmel, California

Prof Amrit L. Goel
Syracuse University

Prof J. F. Hannan
Michigan State University

Prof H. O. Hartley
Texas A & M Foundation

Prof W. M. Hirsch
Courant Institute

Dr Alan J. Hoffman
IBM, Yorktown Heights

Prof John R. Isbell
SUNY, Amherst

Dr J. L. Jain
University of Delhi

Prof J. H. K. Kao
Polytech Institute of New York

Prof W. Kruskal
University of Chicago

Mr S. Kumar
University of Madras

Prof C. E. Lemke
Rensselaer Polytech Institute

Prof Loynes
University of Sheffield, England

Prof Tom Maul
Kowloon, Hong Kong

Prof Steven Nahmias
University of Santa Clara

Prof D. B. Owen
Southern Methodist University

Prof *P. R. Parathasarathy*
Indian Institute of Technology

Prof E. Parzen
Texas A & M University

Prof H. O. Posten
University of Connecticut

Prof R. Remage, Jr.
University of Delaware

Prof Hans *Riedwyl*
University of Berne

Mr David Rosenblatt
Washington, D. C.

Prof M. Rosenblatt
University of *California, San Diego*

Prof Alan J. Rowe
University of Southern California

Prof A. H. Rubenstein
Northwestern University

Prof Thomas L. Saaty
University of Pittsburgh

Dr M. E. Salveson
West Los Angeles

Prof Gary Scudder
University of Minnesota

Prof Edward A. Silver
University of Waterloo, Canada

Prof Rosedith Sitgreaves
Washington, DC

LTC [illegible]
Department of the Army

Prof M. J. Sobel
Georgia Inst of Technology

Prof R. M. Thrall
Rice University

Dr S. Vajda
University of Sussex, England

Prof T. M. Whitin
Wesleyan University

Prof Jacob Wolfowitz
University of South Florida

Prof Max A. Woodbury
Duke University

Prof S. Zacks
SUNY, Binghamton

Dr Israel Zang
Tel-Aviv University

February [illegible]

DATE
FILMED

[illegible]-8[illegible]